SCO ISAM User's Reference

The Santa Cruz Operation, Inc.

Document version: 1.0.0A

Date: 17 July 1990

Table of Contents

1

Introduction

SCO ISAM is a library of routines for managing data. It provides a complete set of tools for a programmer to build custom data management procedures into any type of application.

Data management is key to a wide variety of applications, even those whose main focus is not data. While commercial database management systems (DBMS) are designed as general-purpose data managers, they often cannot be customized enough to meet specific needs. Also, data management may be a small part of an application, and using a DBMS may be a waste of resources, even if it could be interfaced properly.

SCO ISAM uses indexed sequential access methods (ISAM) to provide low-level access to data files. It gives the programmer precise control over how data is handled. The data handling routines can be embedded directly in the flow of a custom application. SCO ISAM is also very fast. The trade-off is the need for the programmer to keep track of the details of file structure when writing programs.

SCO ISAM is based on the data management (ISAM) standard proposed by X/OPEN as a part of the Common Application Environment. Applications written based on the standard are portable across operating systems and hardware.

SCO ISAM Enhancements

The SCO ISAM version of ISAM enhances the X/OPEN standard in the following ways:

- Case sensitivity may be turned off in string searches.
- Routines to generate statistics on the use of ISAM files are provided.
- Alternate, international collating sequences are available.
- Functions are provided to generate unique data values.
- Shared locking is supported.

Organization of this Guide

This guide is organized to give the programmer specific information needed to incorporate ISAM into applications. If you are an experienced programmer, you may want to go directly to Chapter 4, "Programming Overview," which gives you enough information to build an ISAM application quickly. You may also want to refer to Chapter 15, "Reference," which gives complete descriptions of function calls.

The chapters and appendixes in this guide are organized as follows:

- **Chapter 2**, "What is ISAM?," briefly describes ISAM functions.
- **Chapter 3**, "ISAM Concepts," describes concepts common to all ISAM calls.
- **Chapter 4**, "Programming Overview," provides a demonstration of basic programming strategies in ISAM.
- **Chapter 5**, "Data Types," describes the data types that are supported in ISAM.
- **Chapter 6**, "Index Structures," describes the structures that are used to define indexes.
- **Chapter 7**, "Creating an ISAM file," describes the functions and steps used to create a new ISAM data file and primary index.

- **Chapter 8**, "Adding an Index," explains the techniques for adding a secondary index to an ISAM file.
- **Chapter 9**, "Adding Data," explains the techniques for inputing data into an ISAM file.
- **Chapter 10**, "Sequential Data Access," explains the techniques for extracting data in sequential order from an ISAM file.
- **Chapter 11**, "Random Data Access," explains the techniques for locating individual records in an ISAM file.
- **Chapter 12**, "Updating a Record," explains the techniques for changing part or all of the data in a record.
- **Chapter 13**, "Deleting a Record," explains the techniques for removing a record from an ISAM file.
- **Chapter 14**, "Locking," explains the techniques for locking ISAM files and records to exclude access by other users on a multiuser operating systems.
- **Chapter 15**, "Reference," provides the complete syntax for all SCO ISAM function calls.
- **Appendix A**, "The isam.h Header File," lists the contents of the *isam.h* header file.
- **Appendix B**, "The example.h Header File," lists the contents of the *example.h* header file.
- **Appendix C**, "Exception Handling," lists ISAM error codes and diagnostics about the type of error.
- **Appendix D**, "Administering ISAM Databases," describes the utilities for maintaining ISAM databases.

2

What is ISAM?

ISAM stands for Indexed Sequential Access Method. This is a method for storing and retrieving data from a file sequentially or randomly.

An ISAM file in SCO ISAM consists of an index file and a data file. An index identifies the location of each record in the data file based on a key value. As new data is added to an ISAM file, the data is written into the data file and the indexes are updated.

You do not need not to be aware of the internal structure of ISAM files. The function calls take care of the details of maintaining the data and index files. However, you should understand the structure of records and index keys and the other ISAM concepts described in Chapter 3, "ISAM Concepts."

ISAM Functions

SCO ISAM provides the following functions to manipulate ISAM files.

Create and delete data and index files:

isbuild	Create a new ISAM file.
iserase	Delete the specified ISAM file.
isrename	Rename a specified ISAM file.
isaddindex	Add a new index to an existing ISAM file.
isdelindex	Delete specified index from the ISAM file.

Open and close ISAM files:

isopen	Open an ISAM file in specified access and lock modes.
isclose	Close an ISAM file.

Select an alternate index to use with an open ISAM file:

isstart	Select an index and locate a specified record.

Read, write, and update records:

isread	Read a record specified by the direction parameter.
iswrite	Write a record.
iswrcurr	Write a record and set record position and record number.
isrewrite	Rewrite record specified by the current key.
isrewcurr	Rewrite current record and set record number.
isrewrec	Rewrite record specified by record number.
isdelete	Delete a record specified by the current key.
isdelcurr	Delete the current record.
isdelrec	Delete the record specified by record number.

Lock files and records (on multiuser systems only):

islock	Lock an ISAM file.
isunlock	Unlock an ISAM file.
isrelease	Unlock all locked records.
isdelrec	Delete the record based on record number.

Get information about an ISAM file:

isindexinfo Get information about an ISAM file and its keys.

Generate unique data values:

isuniqueid Generate a unique data value.

issetunique Generate a unique incremented data value.

Using ISAM Functions

SCO ISAM consists of a header file and a set of libraries. Any programs using ISAM calls must include the header file and be linked with the library file.

To include the header file, add this line near the beginning of each program file:

```
#include <isam.h>
```

Now the definitions in *isam.h* are available to the code in that program file.

To link to the *isam* library, compile each of your program source files into object files. Then use your link utility to link them with the library file. The command line might look like this:

cc -o outfile file1.o file2.o file3.o, , , -lisam

If you are using a non-international operating system, you must also link your files with the non-international library file:

cc -o outfile file1.o file2.o file3.o, , , -lisam -lintl.stub

The *isam.h* header file is described in Appendix A. Each of the function calls in the *isam* library is fully described in Chapter 15, "Reference." Chapters 7–14 describe program examples that use the functions.

3

ISAM Concepts

This chapter provides an explanation of concepts that are common to all ISAM function calls. Together with the next chapter, "Programming Overview," this chapter gives you enough information to write a simple ISAM application.

The information covered in this chapter includes:

- Fields and records
- Indexes
- Random and sequential access
- Current position
- Record number
- Error handling
- ISAM limits

Fields and Records

Data to be stored using ISAM must be structured into fields and records. A *field* is a category of data, such as a name or an address. A data file can have multiple fields and any number of records. A *record* is one set of data values for all the fields describing a specific subject. In a file that stores address information, for example, each person listed might have a record comprised of data values filling name, address, and phone number fields.

When data is put into table format, fields comprise the columns of the table and records comprise the rows. Unlike a database management program, ISAM does not include a facility for naming fields. Field names, if they are used, must be assigned and maintained by the user's program.

In an ISAM file, each field is assigned a fixed data length depending on the type of data it will hold. A record is stored as the set of the fields strung together. Thus, each record in an ISAM file also has a fixed length that is the sum of the lengths of all the fields in that file. The order and lengths of fields are established when the data file is created.

An ISAM file consists of a fixed header and data records. New records are usually appended to the file. However, if there are any vacant slots available due to previous deletions, new records are stored in these slots.

Indexes

Indexes provide the primary access to the records in ISAM data files. An *index* can be thought of as a list of paired data values and locations, sorted by data value. A calling program can use an index to locate a particular piece of data quickly.

In the simplest case, an index has one part (any single field from an ISAM file) and the data is indexed in ascending order. The data field that is indexed is called the *index key*. When you create an index, an entry is created for each record in the index key. An index entry consists of a record's key value and its location in the data file.

ISAM function calls automatically update each index as records are added, updated, or deleted.

Primary and Secondary Indexes

Each ISAM file has at least one index, called the *primary index* or *primary key*, which is created when the data file is built. The primary index is most useful when it holds unique values so that each record can be identified without ambiguity.

An ISAM file may have more than one index, each of which is defined separately after the file is created. These additional indexes are called *secondary indexes*, and each provides a different route into the data file. An ISAM file can have any number of secondary indexes.

Using Indexes to Access Data

An index is structured as a balanced tree (b+tree). A search for a specific record takes the shortest route through the branches of the tree to the desired key value. New index entries are positioned in the tree so that the index is always maintained in sorted order. This incremental reorganization ensures rapid access to your data.

Each ISAM index provides one sequence for accessing records, regardless of the physical order of the records. The name "indexed sequential access method" comes from this feature. The file appears to be in the order specified by the index.

Because you can create several indexes on a data file, you potentially can locate records by any of the fields or combinations of fields. By opening the same file several times, you can use several indexes at once.

When the ISAM file is opened, the primary index is automatically used to locate records. With the **isstart** call, you can select an alternate index to use when searching the file. The index being used to access data is called the *current index*.

Multipart Indexes

An index can also have multiple parts, combining two or more fields from one ISAM file. Indexing on both last name and first name is an example of a multipart index.

The index parts need not be contiguous; they can be at different locations in a record. Each of the parts can be separately set to ascending or descending order when the index is defined. You can also specify whether case sensitivity should or should not be used during searches of an indexed part with a character data type.

The maximum total length of any index is 120 bytes. A multipart index can include up to nine noncontiguous portions of a record.

Unique Indexes

An index can be defined to accept or not accept duplicate values in the indexed field. When the index is defined to allow no duplicates, it is called a *unique index*, and each record can be uniquely identified by the data in the index key.

Index Compression

Indexes can also be defined to use compression, meaning that the index data is condensed to save disk space. The trade-off is that compression can cause data access to be somewhat slower than with noncompressed indexes. Several types of compression are available. (See Chapter 6, "Index Structures.")

For More Information on Indexes

Chapter 6 describes index structures and how to set up an index on a data file. Chapter 7 describes how the primary index is created when building a new data file. Chapter 8 describes how to add a secondary index to an existing data file.

Random and Sequential Record Access

ISAM provides two methods for retrieving records, *keyed sequential access* and *random access*. Keyed sequential access (or just sequential access) retrieves records in order based on the current index. Random access retrieves records based on the value of a key—you specify a key value for the currently indexed field and the file is searched for a record containing that value. This is useful when you need to locate specific information.

Current Position

An open ISAM file has a *current position* that determines which record is retrieved next when **isread** is used to read the file and to select the next, current, or previous record. (See Chapter 10, "Sequential Data Access," for more on **isread**.)

The **isread** and **isopen** calls set the current position, as do **isstart** (used to select an alternate index and to locate a record) and **iswrcurr** (used to change the contents of a record). The current position is used by **isdelcurr**, **isread**, and **isrewcurr**.

Record Number

The *record number* is an identifier for the record last read, written, or selected with **isstart**. This value is stored in the global variable **isrecnum**.

The value of **isrecnum** can be saved and used for direct access of the previously located record, to update it (using **isrwcurr**) or to delete it (using **isdelrec**). For example, you can read a record that you want to update into a buffer, save the **isrecnum** value in a variable, modify the data in the buffer, and then update the original record as identified by the record number. The actual value of **isrecnum** has little lasting significance.

The following calls update **isrecnum**.

isdelcurr	**isdelete**	**isdelrec**	**isread**
isrewcurr	**isrewrec**	**isrewrite**	**isstart**
iswrcurr	**iswrite**		

Error Handling

It is the responsibility of the programmer to handle error conditions that arise as a result of ISAM function calls. Most ISAM calls generally return a value of zero to indicate successful execution and a value of -1 to indicate an error condition and unsuccessful execution (some ISAM calls return a non-negative integer to indicate an error). When an error occurs, the global integer **iserrno** is set to indicate the nature of the error. The values of **iserrno** are listed in Appendix B.

To make testing for errors easier, a set of macro names corresponding to the values of **iserrno** is defined in *isam.h*. These constant names are also listed in Appendix B.

In addition to **iserrno**, diagnostics about the type of error are provided by two global status characters, **isstat1** and **isstat2**. The **isstat1** variable indicates general information about the status of a function call. The **isstat2** variable provides further information, based on the value of **isstat1**. The values of these two variables are also listed in Appendix B.

Testing Return Values

Generally you should test the return value from each ISAM function call for a -1 error condition. If true, then test for the likely values of **iserrno** or call a general-purpose error testing routine. Once the nature of the error is determined, then perform whatever steps are needed to handle the specific error. For example:

```
if ((fdauthor = isopen("author", ISINOUT+ISEXCLLOCK)) < 0) {
    printf("isopen error %d for author file \n", iserrno);
    exit(1);
}
```

ISAM Limits

The following limitations apply to all ISAM files:

Record length	No limit
Number of records	No limit
Number of ISAM files	10
Number of open ISAM files	20
Index length	120 bytes
Non-contiguous index portions	9
Number of indexes	No limit
Number of locks	2000
Number of values of a duplicate	No limit

4

Programming Overview

This chapter provides an overview of how to write an ISAM application. It demonstrates basic programming strategies and gives examples of the most often used function calls.

Because this chapter is an overview, many details are not explained here. You will still want to read the chapters that follow for more information and to learn about functions that are not covered here. Experienced programmers may just need to read the reference pages in Chapter 15.

The functions of a typical ISAM application are:

- Creating an ISAM file
- Adding data
- Accessing data randomly
- Deleting records
- Updating records
- Adding an index

These components are illustrated with a sample database that stores information about books.

Creating an ISAM file

Before creating an ISAM file, you need to identify the data fields and a primary index. This example creates the following ISAM file:

Field	Data Type	Bytes
ISBN	long	4
Title	char	20
Publisher	char	20
Quantity	long	4
Price	long	4
Total length		**52**

The ISBN field is the International Standard Book Number, which uniquely identifies each book. This is used as the primary index.

To create an ISAM file with SCO ISAM, you fill an index structure with values and then use the **isbuild** function call.

Here is a program fragment that creates an ISAM file with these specifications:

```
#define RECSIZ 52

struct keydesc key ;                    /* index key structure */
int fdbook ;                            /* file descriptor */

key.k_flags = ISNODUPS ;                /* no duplicate values */
key.k_nparts = 1 ;                      /* number of key parts */
key.kpart[0].kp_start = 0 ;             /* position of first byte */
key.kpart[0].kp_leng = LONGSIZE ;       /* length of key part */
key.kpart[0].kp_type = LONGTYPE ;       /* data type of key part */

if ((fdbook = isbuild ("books", RECSIZ, &key, ISINOUT +
ISEXCLLOCK)) < 0)
    /* create the ISAM  file named books */
    /* total record length is RECSIZ */
    /* . . . */
isclose(fdbook);                        /* close the ISAM file */
```

The ISINOUT + ISEXCLLOCK parameter opens the file for input and output in an exclusive lock mode. This is the most common value, but others may be used. See Chapter 6, "Index Structures," for more information.

The index key structure **keydesc** is defined in the *isam.h* header file. That file also holds the definitions of the macros ISNODUPS, LONGSIZE, LONGTYPE, ISINOUT, and ISEXCLLOCK.

Adding Data

To add data to an ISAM file:

1. Open the file with the **isopen** function call.
2. Fill a buffer with the data values.
3. Call the **iswrite** function.
4. Check for errors and close the file when finished.

This program fragment illustrates these steps.

```
int fdbook ;                    /* file descriptor */
char buffer[RECSIZ] ;           /* buffer for new record */
long isbn ;                     /* fields */
char *title ;
char *publisher ;
long quantity ;
long price ;

fdbook = isopen("books", ISINOUT + ISEXCLLOCK);
isbn = 0810462613 ;             /* data values */
title = "Programming in C     " ;
publisher = "Hayden Books         " ;
quantity = 2 ;
price = 2895 ;
stlong(isbn, buffer) ;          /* fill the buffer with values */
strcpy(buffer + 4, title) ;
strcpy(buffer + 24, publisher) ;
stlong(quantity, buffer + 44) ;
stlong(price, buffer + 48) ;
iswrite(fdbook, buffer) ;       /* write the data to the ISAM file */
    /* . . . */
isclose(fdbook) ;               /* close the ISAM file */
```

The **stlong** function converts a long integer to ISAM format. It is defined in the *isam.h* header file.

In place of the **iswrite** function, you could use **iswrcurr** which writes the record and makes the new record the current record. The **iswrite** function does not make the new record the current record. Either function also updates all indexes defined in the file. See Chapter 8, "Updating a Record," for more information.

Accessing Data Randomly

The **isread** function reads a record from an ISAM file into a buffer. It can be used for sequential reads or random reads. To perform a random read:

1. Open the ISAM file.

2. Fill the key field with a search value.

3. Call the **isread** function with the ISEQUAL mode.

The following program fragment illustrates these steps, where the search value is stored in the location in a buffer corresponding to the key field. The record is then read into the same buffer:

```
int fdbook ;                          /* file descriptor */
char buffer[RECSIZ] ;                 /* buffer to hold the record */
long isbn = 0810462613 ;              /* search value */

fdbook = isopen("books", ISINOUT + ISEXCLLOCK) ;
                                      /* open the ISAM file */
stlong(isbn, buffer) ;                /* insert search value */
isread(fdbook, buffer, ISEQUAL) ;     /* read record into buffer */
/* . . . */
isclose(fdbook) ;                     /* close the ISAM file */
```

Once you have the record in the buffer, you can process the values as needed.

Updating Records

In ISAM, updating means rewriting whole records. The steps are:

1. Open the ISAM file.
2. Use **isread** to locate the desired record.
3. Change the data values in the buffer.
4. Call **isrewcurr** to rewrite the current record.

This program fragment changes the book quantity from the previous example.

```
int fdbook ;                    /* file descriptor */
char buffer[RECSIZ] ;           /* buffer to hold selected record */
long isbn ;                     /* index key */
long quantity ;                 /* field to be changed */

fdbook = isopen("books", ISINOUT + ISEXCLLOCK) ;
isbn = 0810462613 ;
quantity = 10 ;
isread(fdbook, buffer, ISEQUAL) ; /* read record into buffer */
stlong(quantity, buffer + 44 ) ; /* insert new value into buffer */
isrewcurr(fdbook, buffer) ;     /* rewrite the current record */
isclose(fdbook) ;               /* close the ISAM file */
```

If the index key used to locate the record is the primary key and does not allow duplicate values (as with this one), a record can be updated directly without reading it first. This is done by filling the buffer with the search value and new data values, and then calling **isrewrite** instead of **isrewcurr**.

The **isrewcurr** function is more often used when the search key allows duplicate values. Because you cannot be sure which record a key will locate, reading the record first gives you the chance to check its contents before updating it.

See Chapter 11, "Random Data Access," for more information.

Deleting Records

Deleting a record is similar to reading a record:

1. Open the ISAM file.
2. Fill the buffer's index field with a search value.
3. Call **isdelete** to delete the matching record.

This program fragment illustrates these steps:

```
int fdbook ;                    /* file descriptor */
char buffer[RECSIZ] ;           /* buffer to hold selected record */
long isbn ;                     /* index key */

fdbook = isopen("books", ISINOUT + ISEXCLLOCK) ;
    /* open the ISAM file */
isbn = 0810462613 ;             /* search value */
stlong(isbn, buffer) ;          /* insert search value into buffer */
isdelete(fdbook, buffer) ;      /* delete the record that matches */
isclose(fdbook) ;               /* close the ISAM file when done */
```

Note that the **isdelete** function does not itself provide an opportunity for confirming the deletion. In an interactive application, you may want to use the **isdelcurr** function instead. That function deletes the most recently read record, which gives you a chance to display the contents and obtain confirmation. See Chapter 12, "Updating a Record," for more information.

Adding an Index

Although you must specify a primary index when creating an ISAM file, you may want to add secondary indexes to access your data by different fields.

Each index that you add is updated each time a record is added, updated, or deleted to the file. Too many indexes can slow processing speeds. To reduce this overhead, you may want to keep only a single primary index and create temporary indexes as needed. The trade-off is the time required to build the temporary index, which can be considerable for a file that contains a large number of data records.

Depending on your application, you might use one of the functions listed in the following table:

Function	Result
isaddindex	Add a secondary index to an existing ISAM file.
isstart	Select an index for use.
isdelindex	Delete a secondary index.

These steps create an index:

1. Open the ISAM file in exclusive mode.
2. Fill a key structure with values.
3. Call **isaddindex** to create the index.

The following program fragment illustrates these steps by adding an index on book title.

```
struct keydesc key ;                    /* index key structure */
int fdbook ;                            /* file descriptor */

fdbook = isopen("books", ISINOUT + ISEXCLLOCK) ;
                                        /* open the ISAM file */
key.k_flags = ISDUPS ;                  /* duplicate values allowed */
key.k_nparts = 1 ;                      /* number of key parts */
key.kpart[0].kp_start = 4 ;             /* position of first byte */
key.kpart[0].kp_leng = 20 ;             /* length of key part */
key.kpart[0].kp_type = CHARTYPE ;       /* data type of key part */
isaddindex (fdbook, &key) ;             /* create the new index */
/* . . . */
isdelindex (fdbook, &key) ;             /* delete the new index */
isclose(fdbook) ;                       /* close the ISAM file */
```

The new index created with **isaddindex** becomes the current index for the open file. To switch to another index, use the **isstart** function.

5

Data Types

This chapter describes the data types that can be used in ISAM files as keys. It also describes how the data types are stored and manipulated.

Each field in an ISAM file is generally assigned one of the allowed data types. However, data types are significant only for index keys. Data of any type can be stored in the non-key fields of a record. It is good practice to keep a specific data type in each field of your ISAM file in case you want to add an index on that field later.

The allowed data types are:

ISAM Data Type	C Data Type	Length
CHARTYPE	char	variable
INTTYPE	short	2 bytes
LONGTYPE	long	4 bytes
FLOATTYPE	float	machine-dependent
DOUBLETYPE	double	machine-dependent

Each ISAM data type is associated with a C-language data type. All number data types also have their lengths assigned in the header file. Note that the lengths of FLOATTYPE and DOUBLETYPE are automatically matched to the corresponding machine data length. The length of character data fields is assigned by the programmer when a data file is created.

CHARTYPE

The CHARTYPE data type is used to store character strings such as names and addresses. Numbers may also be stored in character format if no arithmetic operations are to be performed on them. Zip codes and telephone numbers are examples of such numbers.

Each field that uses CHARTYPE has a fixed length that is assigned when the data file is created. Null termination of character strings is not required. See Chapter 7, "Creating an ISAM File," for examples of how to assign a length to a character field.

INTTYPE and LONGTYPE

The integer data types INTTYPE and LONGTYPE are used to store 2-byte and 4-byte signed integers, respectively. If you are sure the data values for an integer field will be in the range -32768 to +32767, then you can use INTTYPE. Otherwise, use LONGTYPE.

The bytes that make up an integer are always stored in the ISAM file in the order you would expect: most-significant byte first and least-significant byte last. Some machines, however, store integers in the reverse order. Four routines are available to convert integers between ISAM and machine format.

The following two routines convert ISAM integers to machine format:

- **ldint(p)** returns a short integer in machine format. Here p is a char pointer to the first byte of the format INTTYPE.
- **ldlong(p)** returns a long integer in machine format. Here p is a char pointer to the first byte of the format LONGTYPE.

The above routines might be used when ISAM records are written to a temporary storage area for further processing by a C program. Any integers being passed from the buffer to the executing program are first converted to machine format with the **ldint** and **ldlong** routines.

The following two routines convert machine-format integers to ISAM:

- **stint(i,p)** converts a machine-format short integer i to INTTYPE and stores it at location p. Here p is a char pointer to the first byte of INTTYPE.

- **stlong(l,p)** converts a machine-format long integer l to LONGTYPE and stores it at location p. Here p is a char pointer to the first byte of LONGTYPE.

The previous two routines might be used when integers produced or processed by an executing C program are to be written to an ISAM file. The **stint** and **stlong** routines convert the machine-format integers to ISAM format and place them in a buffer in preparation for writing them to the ISAM file.

Integers in ISAM format need not align along word boundaries as must integers in some machine format.

FLOATTYPE and DOUBLETYPE

The data types FLOATTYPE and DOUBLETYPE are used to store single- and double-precision floating-point numbers, respectively.

These ISAM data types correspond to the C-language data types float and double. The length and format of these data types are machine dependent. The definitions of FLOATTYPE and DOUBLETYPE in **isam.h** automatically adjust them to the sizes on the current machine.

On certain machines, floating-point numbers must generally be aligned on word boundaries. ISAM-format numbers need not be. Four routines are available to convert between the unaligned ISAM format and the aligned machine format.

The following two routines convert ISAM floating-point numbers to machine format:

- **ldfloat(p)** returns a float in machine format. Here p is a char pointer to the first byte of the format FLOATTYPE.
- **lddbl(p)** returns a double in machine format. Here p is a char pointer to the first byte of the format DOUBLETYPE.

The above routines might be used when ISAM records are accessed for processing. Any floating-point numbers being passed from the buffer to the executing program must be first converted to machine format with the **ldfloat** and **lddbl** routines.

The following two routines convert machine-format floats and doubles to ISAM:

- **stfloat(f,p)** converts a machine-format float f to FLOATTYPE and stores it at location p. Here p is a char pointer to the first byte of FLOATTYPE.
- **stdbl(d,p)** converts a machine-format double d to DOUBLETYPE and stores it at location p. Here p is a char pointer to the first byte of DOUBLETYPE.

The above two routines might be used when floating-point numbers produced or processed by an executing C program are to be written to an ISAM file. The **stfloat** and **stdbl** routines convert the machine-format numbers to ISAM format and place them in a buffer in preparation for writing them to the ISAM file.

6

Index Structures

Indexing provides the primary access to data in ISAM data files. This chapter describes index structures and how to set up an index on a data file. Chapter 7, "Creating an ISAM File," describes how the primary index is created when building a new data file. Chapter 8, "Adding an Index," describes how to add a secondary index to an existing data file.

ISAM uses two C-program structures, **keydesc** and **keypart**, to specify an index. These structures are defined in the *isam.h* header file.

The first structure, **keydesc**, describes the index as a whole. It has the following members:

```
short k_flags;              /* duplicate and compression flags */
short k_nparts;             /* number of parts in this index */
struct keypart k_part[NPARTS];          /* array of key parts */
```

The second structure, **keypart**, is nested in the first structure and describes each part of an index. It has the following members:

```
short kp_start;      /* offset of first byte of this key part */
short kp_leng;       /* number of bytes in this key part */
short kp_type;       /* data type of this key part */
```

Note that the sum of the lengths of all the parts cannot exceed MAX-KEYSIZE, which is defined as 120 bytes in *isam.h*. The maximum number of parts in a key cannot exceed NPARTS (also defined in *isam.h*).

Index Structure Members

The **keydesc** and **keypart** structures use members as part of their definition in the *isam.h* header file. The following sections describe these members.

k_flags

The k_flags integer is used to indicate whether this index allows duplicate values and/or uses compression of the index to save space. The values that can be used in k_flags are:

Macro Name	Description
ISNODUPS	No duplicates allowed.
ISDUPS	Duplicates allowed.
DCOMPRESS	Compression of duplicates.
LCOMPRESS	Leading compression.
TCOMPRESS	Trailing compression
COMPRESS	All compressions.
ISALL	All compressions and allows duplicates.

These values are defined in the *isam.h* header file. The different values of compression imply different levels of compression, which affect performance to different degrees. Values can be summed to combine options. Note that the last two values are sums of previous values and are provided for convenience.

No default value is defined, so k_flags must be initialized to some value. This means if no compression is desired, then either ISNODUPS or ISDUPS must be used.

k_nparts

The k_nparts integer specifies the number of parts in the index. The number of parts cannot exceed NPARTS, an integer defined in the *isam.h* header file.

k_part

k_part is an array, each member of which has the structure **keypart**. Each member describes one part of the index.

kp_start

The kp_start integer indicates the starting byte of a key part within a record. You must examine the data file definition to determine which byte this is for a particular field. Note that this is the offset into the record, so the first byte of a record is always zero.

kp_leng

The kp_leng integer indicates the number of bytes of a key. For number fields, this is the length of the data type. For character fields, this can be all or part of the character field.

kp_type

The kp_type integer indicates the data type of the field. The data types as defined in the *isam.h* header file are:

Type	Description
CHARTYPE	Character data type.
INTTYPE	Short integer data type.
LONGTYPE	Long integer data type.
DOUBLETYPE	Double-precision floating point.
FLOATTYPE	Single-precision floating point.

The kp_type integer also can indicate options for sorting. The two options are:

Sort Option	Description
ISDESC	Descending sort order.
ISNOCASE	No case sensitivity.

These values are added to the data type to turn on the options. ISDESC can be used with any data type, but ISNOCASE only makes sense for character fields. By default, the index is in ascending order, and the character part of the key (if any) is case sensitive during searches.

7

Creating an ISAM File

This chapter describes how to build a new ISAM file. It describes how to use the **isbuild** function call to construct a database for a bookstore. The example database that is used in this and subsequent chapters is also described here.

When an ISAM file is created, generally part of each record is defined as a primary index. The primary index usually has unique values so records can be identified without ambiguity. Before reading this chapter, you may want to review the general information on indexes in Chapter 3, "ISAM Concepts."

If necessary, the primary index can be defined to allow duplicate values. You can also avoid defining a primary index altogether. See the **isbuild** reference page in Chapter 15 for more information on these options.

If you are using an international operating system and a language other than U.S. English, SCO ISAM can use alternate collating sequences (U.S. ASCII is the default). The sequence to be used is retrieved by the **setlocale** function when SCO ISAM is initialized, and is determined by the language specified in your international environment. For more information on the international environment, see your operating system documentation.

When an ISAM file is created, the current collating sequence is stored in the header. If you later open this file when a different collating sequence is active, an error results.

isbuild creates two files, one for index and the other for data. The **isrename** function lets you rename an ISAM file. The **iserase** function lets you delete an ISAM file. These functions rename and erase both data and index files.

The isbuild Function

The **isbuild** function creates a new ISAM file and leaves it open for data to be written into it. This line from the example program shows how it is typically used:

```
if ((fdbook = isbuild("book", 52, &key, ISINOUT + ISEXCLLOCK)) < 0)
```

The **isbuild** function is called with four arguments:

Argument	Description
"book"	The name of the ISAM file being created. The length of the filename must be 10 characters or less.
52	The number of bytes in a record. This is the sum of the byte count of all of the fields.
&key	The definition of the primary index. This is filled before the call is made.
ISINOUT + ISEXCLLOCK	The lock mode in effect while the file is open is **isbuild**. A value of ISINOUT + ISEXCLLOCK indicates the file is open for input and output, and that it has an exclusive lock so other processes cannot access it.

If the call is successful, it returns the file descriptor of the new ISAM file. The file descriptor is assigned to fdbook in this example.

See the **isbuild** reference page in Chapter 15 for a complete description of this function.

The Example Database

The database used in the examples in this and subsequent chapters consists of two ISAM files: books and authors. The records for the two files look like this:

Books File

Field	Data Type	Bytes
ISBN	long	4
Title	char	30
Publisher	char	30
Quantity	long	4
Price	double	8
Total length		**76**

Authors File

Field	Data Type	Bytes
ISBN	long	4
Author #	long	4
Last Name	char	30
First Name	char	30
Total length		**68**

Description of Example Program

This program uses the **isbuild** function to create the two ISAM files *books* and *authors*. The primary index for both files is the ISBN (International Standard Book Number) field.

The program sequence is:

1. Define the key description for ISBN.
2. Create the *books* file with record length 76.
3. Create the *authors* file with record length 68.
4. Close both files.

Note these features of the program:

- The fields of an ISAM file are not actually given names in the data file. Each record is just a sequence of bytes. It is the programmer's responsibility to keep track of field names.
- The primary index description (called key here) must be filled before calling **isbuild**.
- The primary index has a C structure named **keydesc**, which is defined in *isam.h*.

See Chapter 3, "ISAM Concepts," and Chapter 6, "Index Structures," for more information on indexes.

Program Example

```
/*
*   build.c
*   A sample program to illustrate isbuild.  Builds both "books" and
*   "authors" files. For details on the structure of these files, see
*   "example.h"
*
*
*   isbuild and isopen return a positive number on success. All other
*   calls return 0  on success and -1 on failure. On failure, iserrno
*   is set to indicate the error.
*/

#include     <stdio.h>
#include     "isam.h"
#include     "example.h"              /* This must come after isam.h */

main()
{
    int      fdbooks, fdauthors ;    /* Isam file descriptors */
    long     bookcode, quantity ;

    struct   keydesc key ;

    /* Set up key description for bookcode    */

    key.k_flags              = ISNODUPS ;
    key.k_nparts             = 1 ;
    key.k_part[0].kp_start   = BOOKCDOFF ;
    key.k_part[0].kp_leng    = BOOKCDSIZE ;
    key.k_part[0].kp_type    = LONGTYPE ;

    if ((fdbooks = isbuild("books",BOOKRECSIZE, &key,
                           ISINOUT + ISAUTOLOCK)) < 0)
         printf("\nError building books file -- status = %d", iserrno) ;
    else
         printf("\nBooks file built") ;
    /*
    * bookcode is also a key in "authors". The only difference in the
    * key description is that duplicate values of bookcode are allowed,
    * since one book may have more than one author
    */
    key.k_flags      = ISDUPS ;
```

```
    if ((fdbooks = isbuild("authors",AUTHRECSIZE, &key,
                            ISINOUT + ISAUTOLOCK)) < 0)
            printf("\nError building authors -- status = %d", iserrno) ;
    else
            printf("\nAuthors file built") ;

    isclose(fdbooks) ;
    isclose(fdauthors) ;
}
```

8

Adding an Index

This chapter describes how to add a secondary index to an ISAM file. Remember that the primary index is created when the data file is built using **isbuild**. Secondary indexes may be added at any time to provide multiple avenues of access to a data file. In this example, an index is added so that books may be located by author's first and last names.

Before reading this chapter, you may want to review the general information on indexes in Chapter 3, "ISAM Concepts," and Chapter 6, "Index Structures."

The example program in this chapter uses two ISAM function calls, **isopen** and **isaddindex**.

The isopen Function

The **isopen** function opens an existing data file for use and sets the current position to the first record of the primary key. This line from the example program shows how it is typically used:

```
if ((fdauthor = isopen("author", ISINOUT+ISEXCLLOCK)) < 0)
```

It is called with two arguments:

Argument	Description
"author"	Name of the ISAM data file being opened.
ISINOUT + ISEXCLLOCK	The lock mode in effect while the file is open. A value of ISINOUT + ISEXCLLOCK indicates the file is open for input and output, and that it has an exclusive lock so other processes cannot access it. Other values of the mode parameter are described on the isopen reference page in Chapter 15.

On multiuser operating systems, the file you are indexing must be opened in exclusive lock mode before you can add the index. Although this is not required on single-user operating systems, it is a good idea to use the exclusive lock anyway so the application can later be ported to a multiuser system.

If successful, the **isopen** function returns the file descriptor for the data file. The file descriptor is assigned to fdauthor in this example.

The isaddindex Function

The **isaddindex** function creates an index for an existing ISAM file. All the records currently in the data file are indexed. This line from the example program shows how it is typically used:

```
if (isaddindex(fdauthor, &key) < 0)
```

It is called with two arguments:

Argument	Description
fdauthor	File descriptor of the ISAM file being indexed.
&key	The definition of the new index key.

The general structure for indexes is defined in *isam.h.* You add the particular values to the structure before calling **isaddindex**. If successful, **isaddindex** returns zero.

Program Example

```
/*
*   addindex2.c
*   A sample program to illustrate addition of multi-part indexes. In
*   this program an alternate index is added to "authors". The index
*   consists of two parts - lastname and firstname.
*
*   For details on the structure of the authors file see example.h.
*
*   isbuild and isopen return a non-negative number on success. All
*   other calls return 0 on success and -1 on failure. On failure,
*   iserrno is set to indicate the error.
*/


#include    <stdio.h>
#include    "isam.h"
#include    "example.h"          /* This must come after isam.h*/


main()
{
    int     fdauthors ;          /* Isam file descriptor */
    struct  keydesc key ;
    if ((fdauthors = isopen("authors", ISINOUT + ISEXCLLOCK)) < 0){
            printf("\nError opening authors file - status = %d", iserrno) ;
            exit(1) ;
    }

    /* Set up key description  */

    key.k_flags             = ISDUPS ;
    key.k_nparts            = 2 ;
    key.k_part[0].kp_start  = LNAMEOFF ;
    key.k_part[0].kp_leng   = LNAMESIZE ;
    key.k_part[0].kp_type   = CHARTYPE ;

    key.k_part[1].kp_start  = FNAMEOFF ;
    key.k_part[1].kp_leng   = FNAMESIZE ;
    key.k_part[1].kp_type   = CHARTYPE ;
```

```
    if (isaddindex(fdauthors, &key) < 0)
            printf("\nError adding name index -- status = %d", iserrno) ;
    else
            printf("\nName Index added") ;

    isclose(fdauthors) ;
}
```

9

Adding Data

This chapter describes how to add data to an existing ISAM file. The **iswrite** function is normally used for adding data records to an ISAM file. As records are added, indexes are updated automatically.

The iswrite Function

The **iswrite** function adds new records to an open ISAM file. Before issuing the call, the string of data for a new record should be loaded into a buffer. This line from the example program shows how it is typically used:

```
if (iswrite(fdbook, bookrec) < 0)
```

It is called with two arguments:

Argument	Description
fdbook	File descriptor of the file to which the record is being added. The file descriptor is obtained with **isopen.**
bookrec	A character buffer holding the data to be written into the file.

If successful, **iswrite** adds the new record, updates any indexes defined on the file, and returns zero.

Program Example

```
/*
*   write.c
*   A sample program to illustrate iswrite. Loads data from a sequential
*   delimited file into books file. Assumes that the books file
*   is already built (see build.c).
*
*   The structure of "books" is described in "example.h"
*
*
*   isbuild and isopen return a non-negative number on success. All other
*   calls return 0 on success and -1 on failure. On failure, iserrno is
*   set to indicate the error.
*/
#include    <stdio.h>
#include    <errno.h>
#include    "isam.h"
#include    "example.h"          /*This must come after isam.h*/
{
    FILE    *fp ;                /* Text file containing input data*/
    int     fdbooks,             /* Isfd for books file*/
            reccnt ;
    long    bookcode,            /* Fields in the book file*/
            qty ;
    float   price ;
    char    title[ TITLESIZE ],
            publisher[ PUBLSIZE ],
            booksrec[ BOOKRECSIZE ] ;     /* Record buffer for books */
    extern  int     errno ;

    if ((fp = fopen("books.seq", "r")) == NULL){
            printf("\nError Opening books.seq -- status = %d\n", errno) ;
            exit(1) ;
    }

    if ((fdbooks = isopen("books", ISINOUT + ISAUTOLOCK)) < 0){
            printf("\nError opening books file -- status = %d\n",iserrno) ;
            fclose(fp) ;
            exit(1) ;
    }

    reccnt = 0 ;
    /*
     * Input data is in ascii, each field separated by a comma.
     * Records are terminated by newline. The code assumes that
     * the field lengths in input do not exceed their maximum length
    */
    while(fscanf(fp, "%ld,%[^,],%[^,],%ld,%f",
                &bookcode, title, publisher, &qty, &price) != EOF){
```

```
        /*
         * Convert all integers and floats to machine independent
         * format using stlong and stdbl
         */
        stlong(bookcode, booksrec) ;
        stlong(qty, booksrec + QTYOFF) ;
        stfloat(price, booksrec + PRICEOFF) ;
        strncpy(booksrec + TITLEOFF, title, TITLESIZE) ;
        strncpy(booksrec + PUBLOFF, publisher, PUBLSIZE) ;

        if (iswrite(fdbooks, booksrec) < 0)
            printf("\nError writing books - status=%d bookcode=%ld"
                ,iserrno, bookcode) ;
        else
            reccnt++ ;
    }
    printf("\n %d records written", reccnt) ;

    fclose(fp) ;
    isclose(fdbooks) ;
}
```

10

Sequential Data Access

This chapter describes how to write a program to read an ISAM file sequentially. This is useful when you want to retrieve a portion of the ISAM file in a particular order. The example program here outputs a list of all the books by the publisher.

The order in which the records are read is based on the index currently in use. When a file is opened, the primary index is the default index. You can select an alternate index with the **isstart** ISAM function call.

The isread Function

The **isread** function reads a record into the specified buffer. When doing sequential accesses, you can select the first, last, next, previous, or current record. For random accesses, you can select a record that matches a key value. This line from the example program shows how it is typically used:

```
if (isread(fdbook, bookrec, ISFIRST) < 0)
```

The **isread** function is called with three arguments:

Argument	Description
fdbook	File descriptor of the file to be read. The file must already by open.
bookrec	A buffer into which the record is transferred after the call is executed successfully.
ISFIRST	The mode parameter identifying which record is to be read. Using ISFIRST locates the first record in the file according to the curren index. Other mode values are described on the **isread** reference page in Chapter 15.

If successful, the **isread** function copies the selected data into the record buffer and returns zero.

The isstart Function

When you open an ISAM file, you are automatically started with the primary index. The **isstart** function is generally used to select an alternate index.

The **isstart** function also locates a record in a manner similar to the **isread** function. However, **isstart** does not actually read the record it locates.

The **isstart** function is generally called with five arguments, which are described below. If you are selecting the first, last, next, previous, or current record in the file as in the program example here, then only three arguments are actually useful. All five arguments are used during random searches for a record.

This line from the example program shows how **isstart** is typically used:

```
if (isstart(fdbook, &key, 0, bookrec, ISFIRST) < 0)
```

The five arguments for the **isstart** function are:

Argument	Description
fdbook	File descriptor of the file to be read. This is obtained from a previous **isopen** call.
&key	Structure describing the index to be selected. This is the same structure used to create the index..
length	Number of bytes of the index key to use during a random search for a record.
record	Character buffer holding the key value to be used in a random search for a record.
mode	Parameter that determines which record is selected. Using ISFIRST causes ISAM to position the record pointer just before the first record in the order of the selected index. Other mode values are described on the **isstart** reference page in Chapter 15.

If successful, the **isstart** function returns zero.

Program Example

```
/* readseq.c
*   A sample program to illustrate sequential read, assuming that the
*   file is built and loaded with data (see build.c and write.c).
*   This program reads records sequentially from the books file and
*   prints the bookcode, title and price.
*
*   The structure of "books" file is described in "example.h"
*
*
*   isbuild and isopen return a non-negative number on success. All other
*   calls return 0  on success and -1 on failure. On failure, iserrno
*   is set to indicate the error.
*/

#include    <stdio.h>
#include    "isam.h"
#include    "example.h"          /* This must come after isam.h*/

main()
{
    int     fdbooks , reccnt;
    long    bookcode , stock;
    float   price ;
    char    booksrec[BOOKRECSIZE] ;
    char    title[TITLESIZE + 1] ;
    struct  keydesc key ;

    if ((fdbooks = isopen("books", ISINOUT + ISAUTOLOCK)) < 0){
            printf("\nError opening books file - status = %d\n", iserrno);
            exit(1);
    }
    reccnt = 0 ;
    /*
     *  When a file is open it automatically selects the primary
     *  index as the default index. In our case this is fine. If
     *  you need to access information by any other index you
     *  must use isstart to switch to that index.
     */
    while (isread(fdbooks, booksrec, ISNEXT) == 0){
            price= lddbl(booksrec + PRICEOFF);
            bookcode = ldlong(booksrec + BOOKCDOFF);
            stock = ldlong(booksrec + QTYOFF);
            printf("%ld,%-*.*s,%3.2f,%ld\n", bookcode,TITLESIZE,
                    TITLESIZE, booksrec+TITLEOFF, price, stock);
            reccnt++ ;
    }
    if (reccnt)
            printf("\n %d records read", reccnt);
    else
            printf("\n Books file empty");

    isclose(fdbooks);
}
```

11

Random Data Access

This chapter describes how to read a particular record in a data file based on a key value. This is useful when you want to look up specific information without having to read through all the data.

You select and read a record with the **isread** function. For random access, you specify a particular mode parameter and a key value for an indexed field. For example, you may want to locate records for books that were written by an author whose last name is Norton. To do so, you would put the character string "Norton" in a buffer and then call **isread** with the ISEQUAL mode parameter. The string's position in the buffer should be the same as the position of the last name field in the records. This call gives you the first record containing "Norton" in the last name field. You can then use the ISNEXT mode parameter to access the other books by this author.

Instead of an exact search, you can also do an approximate search. By using the ISGREAT mode parameter in **isread**, for example, you can locate the first record that is greater than the specified key value. This is useful when you want an approximate key match.

Because ISAM records are just a sequence of bytes, you can also select any combination of bytes as your key field. You could specify "Norton" in the last name field and "Peter" in the first name field to locate books by Peter Norton. You can also take any subset of a character field; however, you cannot split the bytes of a number field.

Because ISAM uses indexes to locate records randomly, whatever field you use must have been indexed. If it has not been, use **isaddindex** first to define an index on that field.

The index to be used is selected with the **isstart** ISAM function call. **isstart** can also be used to locate a record by key value. Once you have selected an index, you can use **isread** to read records.

Using isread for Random Access

As described in the previous chapter, the **isread** function is used to read existing records from an ISAM file.

For random accesses, you must specify a key value and use one of the three random access values of the mode parameter: ISEQUAL, ISGREAT, or ISGTEQ. This line from the example program shows how these are typically used:

```
if (isread(fdauthor, author_rec, ISEQUAL) < 0)
```

The three arguments are:

Argument	Description
fdauthor	File descriptor of the file to be read. This is obtained from a previous **isopen** call.
author_rec	A buffer that holds the search value before the call, and that holds the desired record after the call is executed. For random access (as in this example), the search value should be placed in the appropriate byte positions in the record buffer before the call is executed.
mode	Parameter identifying which record is to be read. Using **ISEQUAL** searches the file for an exact match on the data in the record buffer. Other mode values are described on the **isread** reference page in Chapter 15.

If successful, the **isread** function copies the selected data into the record buffer and returns zero.

Program Example

```
/*
*   readrand.c
*   A sample program to illustrate random read. This program assumes that
*   the file is built and loaded with data (see builde.c and write.c).
*   This program reads a record from the books file given the book code
*
*   The structure of "books" is described in "example.h"
*
*
*   isbuild and isopen return a non-negative number on success. All
*   other calls return 0 on success and -1 on failure. On failure,
*   iserrno is set to indicate the error.
*/

#include     <stdio.h>
#include     "isam.h"
#include     "example.h"          /* This must come after isam.h*/

main(argc, argv)
int argc ;
char         *argv[] ;
{
    int      fdbooks ;
    long     bookcode ;
    float    price ;
    char     booksrec[BOOKRECSIZE] ;
    char     title[TITLESIZE + 1] ;
    struct   keydesc key ;

    if ((fdbooks = isopen("books", ISINOUT + ISAUTOLOCK)) < 0){
            printf("\nError opening books file -- status = %d\n", iserrno) ;
            exit(1) ;
    }

    /* Select the title index */

    key.k_flags              = ISNODUPS ;
    key.k_nparts             = 1 ;
    key.k_part[0].kp_start   = BOOKCDOFF ;
    key.k_part[0].kp_leng    = BOOKCDSIZE ;
    key.k_part[0].kp_type    = LONGTYPE ;

    if (argc < 2){
            printf("\nEnter bookcode: ") ;
            scanf("%ld",&bookcode) ;
    }
    else
            sscanf(argv[1],"%ld",&bookcode) ;
```

```
    stlong(bookcode, booksrec + BOOKCDOFF) ;
    if (isstart(fdbooks, &key, 0 ,booksrec, ISEQUAL) < 0)
            printf("\nRecord not found - status = %d\n", iserrno) ;
    else    {
            if(isread(fdbooks, booksrec, ISNEXT) < 0)
                printf("\nError reading books - status= %d\n",iserrno) ;
            else    {
                    price = ldfloat(booksrec + PRICEOFF) ;
                    bookcode = ldlong(booksrec + BOOKCDOFF) ;
                    printf("\n%ld,%-*.*s,%3.2f", bookcode, TITLESIZE,
                            TITLESIZE,booksrec+ TITLEOFF, price) ;
            }
    }

    isclose(fdbooks) ;
}
```

12

Updating a Record

This chapter describes how to change the data in a record. This is useful for keeping a database up to date.

The term used to describe updates in ISAM is *rewrite*. When you update a record in an ISAM file, the function calls actually overwrite the existing record completely. So the general programming method is to copy the record into a buffer, change the data in the buffer, and rewrite the buffer into the ISAM file.

Each of these calls is used somewhat differently to meet different needs. They are described in more detail in the following sections.

The Rewrite Function Calls

Three function calls are available to update records. These are analogous to the three function calls for deleting records.

- The **isrewcurr** function updates the current record with the contents of the record buffer.
- The **isrewrite** function locates a record based on its primary key and updates it with the contents of the record buffer.
- The **isrewrec** function locates a record based on its record number and updates it with the contents of the record buffer.

The isrewcurr Function

The **isrewcurr** function updates the current record. The record must be established as current by an **isread** call. This is how it is to be used:

```
if (isrewcurr(fdbook, bookrec) < 0)
```

The function is called with two arguments:

Argument	Description
fdbook	File descriptor of the book file containing the record to be updated.
bookrec	The buffer containing the new value of the record. These values overwrite the old values in the file.

The isrewrite Function

The **isrewrite** function is the least frequently used call for updating. It can be used to update a record for which you know the primary key value, and for which duplicates are not allowed.

The **isrewrite** call locates and overwrites the record in one step. This line from the program example shows how it is used:

```
if (isrewrite(fdbook, bookrec) < 0)
```

The function is called with two arguments:

Argument	Description
fdbook	File descriptor of the book file containing the record to be updated.
bookrec	A character buffer containing the new version of the book record and the primary key of the record to be updated.

The record buffer should first be loaded with a complete record that contains the new data. If you are making small changes, it is easier to read the old record into the buffer first, then make the changes. Be sure the correct primary key value is entered in the correct byte positions for the primary key. Then call **isrewrite** to rewrite the matching record with the contents of the buffer. If successful, the function returns zero.

The isrewrec Function

The **isrewrec** function is used to update a record for which you know the record number. This call is used instead of **isrewrite** when the file's primary index allows duplicates, or when you are using a secondary index.

The record number is obtained from the global variable **isrecnum**. This variable holds the record number of the last record read or written, or the record selected with **isstart**. So generally, to rewrite a previously located record, you read the record contents into a buffer, save the **isrecnum** value in a variable, modify the buffer with new data values, and then update the record using the **isrewrec** call.

You can perform other functions before updating the record, such as verifying the record's contents. However, your program should not select another record for the same open file before using the stored record number. The actual value of the record number is of little lasting significance.

This line shows how **isrewrec** might be used:

```
isrewrec(fdbook, bookrecnum, bookrec)
```

Its three arguments are:

Argument	Description
fdbook	File descriptor of the file containing the record to be updated.
bookrecnum	Variable holding the record number of the record to be updated.
bookrec	A character buffer containing the new version of the book record.

Program Example

```
/*
*   rewrite.c
*   A sample to illustrate isrewcurr, assuming that the books file
*   is built and loaded with data (see build.c and write.c). The
*   structure of the books file is described in examples.h. The program
*   reads a specified books record and rewrites it with the quantity
*   halved. It then reads back the changed record and prints the
*   contents.
*
*   isbuild and isopen return a positive number on success. All other
*   calls return 0  on success and -1 on failure. On failure, iserrno
*   is set to indicate the error.
*/

#include     <stdio.h>
#include     "isam.h"
#include     "example.h"      /* This must come after isam.h*/

main(argc, argv)
int argc ;
char         *argv[] ;
{
     int     fdbooks ;        /* Isam file descriptor for books file*/
     long    bookcode, quantity ;
     float   price ;
     char    booksrec[BOOKRECSIZE + 1] ;

     if ((fdbooks = isopen("books", ISINOUT + ISAUTOLOCK)) < 0){
             printf("\nError opening books file — status = %d\n", iserrno) ;
             exit(1) ;
     }
     /*
      * Retrieve the book record by bookcode. If sucessful change quantity
      * and rewrite the record
      */

     if (argc < 2){
             printf("\nEnter bookcode: ") ;
             scanf("%ld",&bookcode) ;
     }
     else
             sscanf(argv[1],"%ld",&bookcode) ;

     stlong(bookcode, booksrec) ;

     if (isread(fdbooks, booksrec, ISEQUAL) == 0){
             quantity = ldlong(booksrec + QTYOFF) ;
             price = ldfloat(booksrec + PRICEOFF) ;
             printf("\nOld Rec: %-*.*s,%ld,%3.2f", TITLESIZE, TITLESIZE,
                                booksrec + TITLEOFF, quantity,price) ;

             quantity /= 2 ;
             stlong(quantity, booksrec + QTYOFF) ;
```

```
            if (isrewcurr(fdbooks, booksrec) < 0)
                    printf("\nError rewriting record -- status =%d\n",
                    iserrno) ;
            else
                if (isread (fdbooks, booksrec, ISCURR) == 0){
                    quantity = ldlong(booksrec + QTYOFF) ;
                    price = ldfloat(booksrec + PRICEOFF) ;
                    printf("\nNew Rec: %-*.*s,%ld,%3.2f",TITLESIZE,
                            TITLESIZE, booksrec + TITLEOFF,
                            quantity, price) ;
                }
                else
                    printf("\nError reading record -- status =%d\n",
                            iserrno) ;

    }
    else
            printf("\nError reading books -- status = %d\n", iserrno) ;

    isclose(fdbooks) ;
}
```

13

Deleting a Record

This chapter describes how to delete a record from an ISAM file. This is useful when you have obsolete records you want to remove.

Generally, you locate the record you want to delete with a random access search of the ISAM file. Such searches are described in more detail in Chapter 11, "Random Data Access."

The Delete Function Calls

Three function calls are available to delete records. These are analogous to the three rewrite call described in the previous chapter.

- The **isdelcurr** function deletes the current record.
- The **isdelete** function locates a record based on its primary index key and deletes it.
- The **isdelrec** function locates a record based on its record number and deletes it.

Each of these calls is used somewhat differently to meet different needs. They are described in more detail in the following sections.

The isdelcurr Function

The **isdelcurr** function deletes the current record. This call is used instead of **isdelete** when the file's primary index allows duplicates, or when you are using a secondary index.

An **isread** call is issued prior to using **isdelcurr** to make the desired record the current record. Because this function operates on whatever is the current record, be sure to delete the record before selecting another record. If you must perform other reads before deleting, then use the **isdelrec** function instead.

This line from the program example shows how **isdelcurr** is used:

```
if (isdelcurr(fdauthor) < 0)
```

Its single argument is:

Argument	Description
fdauthor	File descriptor of the file containing the record to be deleted.

Because the **isdelcurr** function deletes the current record no matter what it is, it is a good idea for your program to check the record's contents before issuing the delete call.

The isdelete Function

The **isdelete** function is the least frequently used call for deleting. It can be used to delete a record for which you know the primary key value, when the file's primary index does not allow duplicates.

The **isdelete** call locates and deletes the record in one step. This line shows how it might be used:

```
if (isdelete(fdbook, bookrec) < 0)
```

The function is called with two arguments:

Argument	Description
fdbook	File descriptor of the book file containing the record to be deleted.
bookrec	A character buffer containing the primary key of the record to be deleted.

The record buffer should first be loaded with the key value in the correct byte positions for the primary key. Then **isdelete** is called to delete the matching record. If successful, the function returns zero.

The isdelrec Function

The **isdelrec** function is used to delete a record for which you know the record number. This call is used instead of **isdelete** when the file's primary index allows duplicates, or when you are using a secondary index.

The record number is obtained from the global variable **isrecnum**. This variable holds the record number of the last record read or written, or the record selected with **isstart**. So generally you read the record you want to delete into a buffer, save the **isrecnum** value in a variable, and then delete the record using the **isdelrec** call.

You can perform other functions before deleting the record, such as verifying the record's contents.

This line shows how it might be used:

```
isdelrec(fdbook, bookrecnum)
```

Its two arguments are:

Argument	Description
fdbook	File descriptor of the file containing the record to be deleted.
bookrecnum	Variable holding the record number of the record to be deleted.

Program Example

```
/*
*   delete.c
*   A sample program to illustrate isdelete.  Assumes that the books file
*   is built and loaded with data (see builde.c and write.c). The
*   structure of the books file is defined in "examples.h".
*
*
*   isbuild and isopen return a positive number on success. All other
*   calls return 0  on success and -1 on failure. On failure, iserrno
*   Most of the isam calls return 0 on success and -1 on failure. On
*   failure iserrno is set to indicate the error.
*/

#include    <stdio.h>
#include    "isam.h"
#include    "example.h"      /* This must come after isam.h*/

main(argc, argv)
int argc;
char        *argv[];
{
    int     fdbooks;         /* Isam file descriptor for books file*/
    long    bookcode, quantity;
    char    booksrec[BOOKRECSIZE];

    if ((fdbooks = isopen("books", ISINOUT + ISAUTOLOCK)) < 0){
            printf("\nError opening books file - status = %d\n", iserrno);
            exit(1);
    }
    /*
     * Retrieve the book record by bookcode. If sucessful change quantity
     * and rewrite the record
     */

    if (argc < 2){
            printf("\nEnter bookcode: ");
            scanf("%ld",&bookcode);
    }
    else
            sscanf(argv[1],"%ld",&bookcode);

    stlong(bookcode, booksrec);

    if (isdelete(fdbooks, booksrec) < 0)
            printf("\nError deleting record -- status =%d", iserrno);
    else
            printf("\nDeleted book with bookcode = %d", bookcode);

    isclose(fdbooks);
}
```

14

Locking

Data in ISAM files on multiuser operating systems can be locked so other processes cannot alter it. This is useful when an operation is in progress and the user does not want the data changed until the operation is complete. One example is an airline reservation system, where a clerk might lock a record for an available seat while the customer makes a decision.

On single-user operating systems, the lock functions described here have no effect. However, it is useful to include them so that applications may run correctly under multi-user systems.

Types of Locking

You can lock an entire file or individual records in a file. You have three options for locking entire files:

Option	Description
exclusive lock	Other processes cannot read or write the file, or obtain locks on it, until it is closed.
manual lock	Other processes can read the file, but they cannot write to the file or obtain locks on it. When you specify a manual lock at the file level, you can explicitly lock the file later with an **islock** call and unlock it with an **isunlock** call.
shared lock	Other processes can read the file and acquire shared locks on it, but they cannot change the data or lock the file exclusively until it is closed.

You also have three options for locking individual records:

Option	Description
automatic locking	A single record is automatically locked when it is read, and unlocked just after the next ISAM call is completed. Other processes cannot read, write, or lock the record until it is unlocked.
manual locking	Each record is explicitly locked and unlocked with function calls. Any number of records can be locked, and they may remain locked for as long as needed. Other processes cannot write or lock the record until it is unlocked.
shared locking	A single record is locked and unlocked explicitly. Other processes can read the record and acquire shared locks on it, but they cannot change or obtain exclusive locks on it until it is unlocked.

The methods for using each of these options are described in the following sections.

Exclusive File Locking

Exclusive file locking is the most restrictive: other processes have absolutely no access to the ISAM file. Exclusive file locking is not normally needed for individual record processing or making backups. It is usually used when adding or deleting an index, or when you are writing a large number of records. When you are planning to update more than 10% of the data in the file, it is a good idea to lock the file in this mode.

Exclusive file locking can only be enacted when the file is opened with the **isopen** function call (or **isbuild** if the file is being created). To apply an exclusive file lock, add ISEXCLLOCK to the mode parameter of **isopen** (or **isbuild**). The lock remains in effect until the file is closed with the **isclose** function. For example:

```
bookfd = isopen("books", ISEXCLLOCK+ISINOUT);
[processing steps here]
isclose(bookfd);
```

Manual File Locking

Manual file locking is less restrictive than exclusive file locking. This allows you to lock the entire file for brief periods. When it is locked, it is similar to an exclusive lock. All the records in the file are locked by the current process and are only available to other processes for reading. A file can be locked and unlocked repeatedly while it remains open.

Manual file locking is a two-step process. First, the **isopen** (or **isbuild**) function call that opens the file must include ISMANULOCK in its mode parameter. This enables manual locking but does not actually lock the file. The file is then locked as needed with an **islock** function call, which locks the entire file. It can be unlocked with an **isunlock** function call.

For example:

```
fdbook = isopen("books", ISMANULOCK+ISINOUT);
/* . . . */
/* The file may be accessed by other users */
/* . . . */
islock(fdbook);
/* . . . */
/* Others may not open, read or write this file */
/* . . . */
isunlock(fdbook);
/* . . . */
/* The file may be accessed by others */
```

Shared File Locking

Shared file locking is useful when a user wants to read a file and be guaranteed that the data will not change while the read is in progress. Any number of processes can hold shared locks on a file at one time.

Shared file locking can only be enacted when the file is opened with the **isopen** function call. To apply a shared file lock, add ISSHARLOCK to the mode parameter of **isopen** (or **isbuild**). The lock remains in effect until the file is closed with the **isclose** function. For example:

```
bookfd = isopen("books", ISSHARLOCK+ISINOUT);
[processing steps here]
isclose(bookfd);
```

Automatic Record Locking

Automatic record locking handles the locking chores for single records. Each record is automatically locked just before being read by **isread**, and unlocked just after the next function call. This method is useful when only one record needs to be locked at a time.

Automatic record locking is enabled when the file is opened with the **isopen** function call (or **isbuild** if the file is being created). To enable automatic record locking, add ISAUTOLOCK to the mode parameter of **isopen** (or **isbuild**). As each record is accessed or updated, it is locked automatically. When another record is accessed, the current record is unlocked and the new record is locked.

```
bookfd = isopen("books", ISAUTOLOCK+ISINOUT);
isread(bookfd, recordbuff, ISNEXT);  /* record is locked */
isrewcurr(bookfd, recordbuff);       /* record is unlocked */
isclose(bookfd);
```

Manual Record Locking

Manual record locking lets you lock individual records at will. This is useful when you want to lock more than one record at a time, or when you want precise control over when records are locked and unlocked.

Manual record locking is a two-step process. First, the **isopen** (or **isbuild**) function call that opens the file must include ISMANULOCK in its mode parameter. This enables manual record locking but does not actually lock any records. A record is then locked by adding ISLOCK to the mode parameter of the **isread** call that reads the record.

All records read using the ISLOCK mode remain locked until an **isrelease** function call is issued, which unlocks all locked records. You cannot unlock one out of several locked records.

For example:

```
bookfd = isopen("books", ISMANULOCK+ISINOUT);
isread(bookfd, record1, ISNEXT+ISLOCK); /* record1 locked */
isread(bookfd, record2, ISNEXT+ISLOCK); /* record2 locked */
isread(bookfd, record3, ISNEXT+ISLOCK); /* record3 locked */
isrelease(bookfd);                      /* all records unlocked */
isclose(bookfd);
```

Shared Record Locking

Shared record locking also lets you lock individual records at will. This is useful when you want to read a record and be guaranteed that the data will not be changed while you are reading it.

Shared record locking, like manual record locking, is a two-step process. First, the **isopen** (or **isbuild**) function call that opens the file must include ISMANULOCK in its mode parameter. This enables shared record locking but does not actually lock any records. A record is then locked by adding ISSLOCK to the mode parameter of the **isread** call that reads the record. Note the similarity between the ISLOCK and ISSLOCK flags.

All records read using the ISSLOCK mode remain locked until an **isrelease** function call is issued, which unlocks all locked records. You cannot unlock one out of several locked records.

For example:

```
bookfd = isopen("books", ISMANULOCK+ISINOUT);
isread(bookfd, record1, ISNEXT+ISSLOCK);     /* record1 locked */
isread(bookfd, record2, ISNEXT+ISSLOCK);     /* record2 locked */
isread(bookfd, record3, ISNEXT+ISSLOCK);     /* record3 locked */
isrelease(bookfd);                        /* all records unlocked */
isclose(bookfd);
```

15

Reference

This chapter contains reference pages for all ISAM function calls. Each reference page provides the syntax, arguments, description, and examples for one call. Room is left for notes.

isaddindex

Add an index to an ISAM file.

Syntax:

```
isaddindex (isfd, keydesc)
int                 isfd;
struct keydesc      *keydesc;
```

Description:

The **isaddindex** routine builds an index for an open ISAM file. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor of the ISAM data file being indexed.
keydesc	Structure defining the index (see Chapter 6, ''Index Structures'').

The **isaddindex** routine executes only if the file has been opened in exclusive mode.

The number of parts of the index cannot exceed NPARTS. The sum of the lengths of parts of a key may not exceed MAXKEYSIZE. NPARTS and MAXKEYSIZE are defined in the *isam.h* header file.

An ISAM file can have any number of indexes built on it.

This call returns an error if the index is defined not to allow duplicates and the data contains duplicates.

Example

```
isaddindex(bookfd, &key) ;
```

See Also **isdelindex**
Chapter 6, "Index Structures"
Chapter 8, "Adding an Index"

isbuild

Create an ISAM file.

Syntax:

```
isbuild(filename, recordlength, keydesc, mode)
char            *filename;
int              recordlength;
struct keydesc  *keydesc;
int              mode;
```

Description:

The **isbuild** function creates a new ISAM file. It creates and initializes the necessary files on the disk to hold the data and index files. You can use the following arguments with this routine:

Argument	Description
filename	Null-terminated character string identifying the name of data file being created. Maximum of 10 characters for the filename. The filename extension should not be specified.
recordlength	Number of bytes in a record. Equals the sum of the field lengths.
keydesc	Pointer to the structure defining the primary index of the file.
mode	Access mode in effect while the file is open as a result of **isbuild**. Arithmetic sum of a lock mode value and a read/write mode value. The values are described later in this section.

If **isbuild** executes successfully, it returns a file descriptor of the new ISAM file. The file remains open for further processing. Use the **isclose** function to close the file.

The **isbuild** function must include an index structure for the primary index. However, if you set k_nparts in the structure **keydesc** to zero, then there is effectively no primary index. Additional indexes may be added with the **isaddindex** function.

The mode parameter specifies the lock mode and the read/write mode in effect while the file is open as a result of **isbuild**. The value of the mode parameter is the arithmetic sum of a lock mode value and a read/write mode value.

The lock mode values are defined in *isam.h* as:

Lock Mode	Description
ISAUTOLOCK	automatic record lock
ISMANULOCK	manual lock
ISEXCLLOCK	exclusive file lock

These modes have the following effects:

- When ISAUTOLOCK is used, each record is locked when it is accessed for read, write, delete, or update, and unlocked after the next function call is made.
- When ISMANULOCK is used, you can perform both optional file locking and record locking. Files may be locked and unlocked multiple times by using the **islock** and **isunlock** functions. You can perform record-locking by **isread** with an ISLOCK option. Locked records may be released with an **isrelease** call.
- When ISEXCLLOCK is used, the entire file is locked. Other users cannot read, write, or open the file. The file is unlocked when it is closed with **isclose**.

The read/write mode values are defined in *isam.h* as:

Build Option	Description
ISOUTPUT	open for writing only
ISINOUT	open for reading and writing

On success, **isbuild** returns a non-negative value. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

This example creates an ISAM file named *books*, with record length of 52 and a primary key contained in *key*.

```
isbuild("books", 52, &key, ISINOUT + ISEXCLLOCK) ;
```

See Also **iserase, isrename, isopen**
Chapter 7, "Creating an ISAM File"

isclose

Close an ISAM file.

Syntax:

```
isclose( isfd )
int      isfd;
```

Description:

The **isclose** function is used to close an ISAM file that was opened with **isopen** or **isbuild**. You can use the following argument with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file to be closed.

All locks on the file are released when it is closed.

■ **IMPORTANT:** ISAM files must be closed when you are through with them. Not doing so may corrupt files in certain environments.

On success, **isclose** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isclose( bookfd ) ;
```

See Also: **isbuild, isopen**

isdelcurr

Delete the current record.

Syntax:

```
isdelcurr(isfd)
int          isfd;
```

Description:

The **isdelcurr** function deletes the current record in an ISAM file. It differs from **isdelete**, which deletes a record identified by a key value. You can use the following argument with this routine:

Argument	Description
isfd	File descriptor identifying a currently open ISAM file.

All index entries associated with the current record are also deleted.

You must open the file with the ISINOUT option. Otherwise the operation fails. On success, **isdelcurr** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isdelcurr(bookfd) ;
```

See Also: **isdelete**, **isdelrec**
Chapter 13, "Deleting a Record"

isdelete

Delete record specified by primary key.

Syntax:

```
isdelete( isfd, record )
int             isfd;
char            *record;
```

Description:

The **isdelete** function deletes a record from a file using a key value. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying a currently open ISAM file.
record	Search value pertaining to the current key of the record to be deleted.

If duplicate primary keys are allowed, then **isdelete** should not be used and the **isread** and **isdelcurr** functions should be used instead.

Any index entries associated with the deleted record are also deleted.

The **isrecnum** pointer is set to the record just deleted, and the current record position is unchanged.

You must open the file with the ISINOUT option. Otherwise, the operation fails. On success, **isdelete** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isdelete(bookfd, buffer) ;
```

See Also: **isdelcurr, isdelrec**
Chapter 13, "Deleting a Record"

isdelindex

Delete an index.

Syntax:

```
isdelindex(isfd, keydesc)
int                 isfd;
struct keydesc      *keydesc;
```

Description:

The **isdelindex** function deletes an existing index from an ISAM file. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file whose index is being deleted.
keydesc	Name of the structure defining the index to be deleted.

Any index may be removed except the primary index.

The file must be opened in exclusive mode or the operation fails. On success, **isdelindex** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isdelindex(bookfd, &key) ;
```

See Also: **isaddindex**

isdelrec

Delete record specified by record number.

Syntax:

```
isdelrec(isfd, recnum)
int          isfd;
long         recnum;
```

Description:

The **isdelrec** function deletes a record based on its record number. The record number is a previously obtained **isrecnum** value.

Argument	Description
isfd	File descriptor identifying the ISAM file containing the record to be deleted.
recnum	The record number of the record to be deleted.

The **isdelrec** function sets the **isrecnum** pointer to **recnum**, and leaves the current record position unchanged.

On success, **isdelrec** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isdelrec(bookfd, isrecnum) ;
```

See Also: **isdelete, isdelcurr**
Chapter 13, "Deleting a Record"

iserase

Delete an entire ISAM file.

Syntax: **iserase(filename)**
char *filename;

Description:

The **iserase** function deletes an entire ISAM file from the disk. Both the index and data files are deleted. If the files are not found, an error is returned and **iserrno** is set to -1. Use this function with great care.

You can use the following argument with this routine:

Argument	Description
filename	Null-terminated character string identifying the name of the ISAM file to be deleted.

Example:

```
iserase("books") ;
```

See Also: **isbuild**

isindexinfo

Obtain information about an ISAM file.

Syntax:

```
isindexinfo(isfd, buffer, number)
int                              isfd;
struct { keydesc | dictinfo }*buffer;
int                              number;
```

Description:

The information returned by the **isindexinfo** call can be either dictionary information or index information, depending on the **number** parameter. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file for which information is to be obtained.
buffer	Name of the buffer with structure **keydesc** or **dictinfo** into which the information is placed.
keydesc	Structure defined in *isam.h* to hold index information. Use only if **number** is nonzero.
dictinfo	Structure defined in *isam.h* to hold file dictionary information. Use only if **number** is zero.
number	Non-negative integer determining what information is obtained. If zero, then the file's dictionary information is put into the buffer. Otherwise, information about the index with that number is put into the buffer.

Dictionary information about a file includes the number of indexes defined, the record size, the primary index size, and the number of records in the file. Dictionary information is placed in the buffer by the call when the **number** parameter is set to zero. The buffer should use the **dictinfo** structure defined in *isam.h* as:

```
struct dictinfo {
short di_nkeys ;     /* number of keys defined     */
short di_recsize ;   /* data record size           */
short di_idxsize ;   /* index key size             */
long di_nrecords ;   /* number of records in file  */
};
```

Index information includes flags (duplicates, compression), number of parts in the key, and the length, type, and location of each key part. Index information is placed in the buffer by the call when the number parameter is set to the number of the index. The buffer should use the **keydesc** structure defined in *isam.h* as:

```
struct keypart {
    short kp_start ;         /* starting byte of key part */
    short kp_leng ;          /* length in bytes           */
    short kp_type ;          /* type of key part          */
};

struct keydesc {
    short k_flags ;          /* flags                     */
    short k_nparts ;         /* number of parts in key    */
    struct keypart k_part[NPARTS] ; /* each key part */
};
```

The number of an index can change when indexes are added or deleted. Use the information in **dictinfo** about the number of indexes to check all indexes.

On success, **isindexinfo** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

This example obtains information about the ISAM file:

```
struct dictinfo dictbuffer
isindexinfo(bookfd, dictbuffer, 0) ;
```

This example obtains information about the second index:

```
struct keydesc keybuffer
isindexinfo(bookfd, keybuffer, 2) ;
```

See Also: **isbuild, isaddindex**

islock

Lock an ISAM file.

Syntax: **islock(isfd)**
int isfd;

Description:

The **islock** function locks an entire ISAM file so that other processes cannot write to the file. You can use the following argument with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file to be locked.

For a file to be locked with **islock**, the file must have been opened using the ISMANULOCK value of the mode parameter in **isopen** or **isbuild**.

A file locked with **islock** can be unlocked with the **isunlock** function. On success, **islock** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
islock(bookfd) ;
```

See Also: **isunlock**
Chapter 14, "Locking"

isopen

Open an ISAM file.

Syntax:

```
isopen(filename, mode)
char          *filename;
int           mode;
```

Description:

The **isopen** function opens a data file for processing. The call returns the file descriptor that is to be used by subsequent ISAM calls on that file. Use the **isclose** function to close the file.

You can use the following arguments with this routine:

Argument	Description
filename	Null-terminated character string identifying the name of the ISAM file to be opened.
mode	Access mode in effect while the file is open. Arithmetic sum of a lock mode value and a read/write mode value. The values are described below.

The current index is set to the primary key.

The current record pointer is set to the first record in the file based on the order in the primary index. It can be set to the first record in another index by using the **isstart** function.

The mode parameter specifies the lock mode and the read/write mode in effect while the file is open. The value of the mode parameter is the arithmetic sum of a lock mode value and a read/write mode value.

The lock mode values are defined in *isam.h* as:

Lock Mode	Description
ISAUTOLOCK	automatic record lock
ISMANULOCK	manual record lock
ISEXCLLOCK	exclusive isam file lock
ISSHARLOCK	shared isam file lock

These modes have the following effects:

- When ISAUTOLOCK is used, each record is locked when it is read, and unlocked after the next function call is made.
- When ISMANULOCK is used, the user must manually lock and unlock each record to perform any operation on that record. A record can be locked by using the ISLOCK mode in the **isread** function call, and unlocked with the **isrelease** function call.
- When ISEXCLLOCK is used, the entire file is locked. Other users can neither read from nor to the file. The file is automatically unlocked when it is closed with **isclose.**
- When ISSHARLOCK is used, the entire file is locked with a shared lock. Other users can read the file and place shared locks on it but may not change it. The file is automatically unlocked when it is closed with **isclose.**

The read/write mode values are defined in *isam.h* as:

Access Modes	Description
ISINPUT	open for reading only
ISOUTPUT	open for writing only
ISINOUT	open for reading and writing

On success, **isopen** returns a non-negative integer. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isopen( "books", ISAUTOLOCK + ISINOUT ) ;
```

See Also: **isbuild, isclose**
Chapter 14, "Locking"

isread

Read records in an ISAM file.

Syntax:

```
isread(isfd, record, mode)
int         isfd;
char        *record;
int         mode;
```

Description:

The **isread** function reads a record from an ISAM file into a buffer for further processing. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file to be read. The file must already be open.
record	Buffer holding the record after the call is executed. For random accesses, record is filled with the search value before the call is executed.
mode	Integer identifying which record is read and whether random or sequential access is used. Also can be used to lock the record manually. The possible values are described below.

If **isread** executes successfully, the current record position pointer and **isrecnum** are set to the record just read.

The mode parameter determines which record is read. The possible values depend on whether you want sequential or random access.

The values of mode for sequential access are specified in the *isam.h* header file as:

Read Mode	Description
ISFIRST	first record
ISLAST	last record
ISNEXT	next record
ISPREV	previous record
ISCURR	current record

During random access, the function compares a value stored in the record parameter to the corresponding data in the records. The mode parameter determines the comparison logic as follows:

Read Mode	Description
ISEQUAL	record equal to search value
ISGREAT	record greater than search value
ISGTEQ	record greater than or equal to search value

The search value is placed in the appropriate byte positions in the record buffer. The record that first satisfies the condition (in the order of the index currently in use) is then copied to the record buffer.

A special use of **isread** allows access by a previously set **isrecnum**. First, **isstart** is called with the k_nparts member of the **keydesc** parameter set to zero. This turns off the primary index and causes the file to be read in physical order. Then **isread** is called with the mode parameter set to ISEQUAL. This causes **isread** to read the record located at the **isrecnum** position.

The record to be read can also be locked by specifying the ISLOCK option for manual locks or ISSLOCK for shared locks. The lock is effective only if the **isopen** function was called with a mode parameter that included ISMANULOCK. The record stays locked until an **isrelease** call is issued on the file.

On success, **isread** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

This example reads a record that matches the search value contained in the buffer:

```
isread( bookfd, buffer, ISEQUAL ) ;
```

See Also: **isstart**
Chapter 10, "Sequential Data Access"
Chapter 11, "Random Data Access"

isrelease

Unlocks all manually locked records in a file.

Syntax:

```
isrelease(isfd)
int          isfd;
```

Description:

The **isrelease** function is used to unlock all records that have been manually locked in a file. You can use the following argument with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file whose records are being unlocked.

A record is manually locked when:

- The data file was opened with a mode parameter that included ISMANULOCK, and
- The **isread** function call that read the record included ISLOCK in its mode parameter.

On success, **isrelease** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isrelease( bookfd ) ;
```

See Also: **isread**, **islock**
Chapter 14, "Locking"

isrename

Rename an ISAM file.

Syntax:

```
isrename(oldname, newname)
char        *oldname;
char        *newname;
```

Description:

The **isrename** function renames both the data and index files, specified by oldname to newname. You can use the following arguments with this routine:

Argument	Description
oldname	Null-terminated string identifying the existing ISAM file that is to be renamed.
newname	Null-terminated string that is the new name of the file.

When you use **isrename**, both the old and new filenames must be specified without any extensions and both files must be in the same directory. Otherwise, the operation can not succeed.

If the file with the oldname does not exist or if the file with newname already exists, the operation can not succeed.

On success, **isrename** returns a positive integer. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

This example changes the name of the *books* file to *titles*:

```
isrename( "books", "titles" ) ;
```

isrewcurr

Rewrite the current record.

Syntax:

```
isrewcurr(isfd, record)
int         isfd;
char        *record;
```

Description:

The **isrewcurr** function overwrites the current record in an ISAM file. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file containing the record to be updated.
record	Character buffer whose contents are written into the file.

Each index defined on the data file, including the primary index, is also updated as needed. Unchanged keys are not rewritten.

The record buffer should be loaded with the new data before the call is issued. Care must be taken to ensure that fields are located properly in the buffer.

On return, **isrecnum** is set to the rewritten record and the current record position is unchanged. On success, **isrewcurr** returns a positive integer. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isrewcurr( bookfd, buffer ) ;
```

See Also: **isrewrite, isrewrec**
Chapter 12, "Updating a Record"

isrewrec

Rewrite the record indicated by record number.

Syntax:

```
isrewrec(isfd, recnum, record)
int     isfd;
long    recnum;
char    *record;
```

Description:

The **isrewrec** function rewrites a record specified by its record number. The record number **recnum** is a previously obtained **isrecnum** value. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file containing the record to be updated.
recnum	Record number of the record to be updated.
record	Character buffer whose contents are written into the file.

The record buffer should be loaded with the new data before the call is issued. Care must be taken to ensure that fields are located properly in the buffer.

Each index defined on the data file, including the primary index, is updated as needed. On return, **isrecnum** is set to **recnum**, and the current record position is unchanged.

On success, **isrewrec** returns a positive integer. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isrewrec( bookfd, isrecnum, buffer ) ;
```

See Also: **isrewrite, isrewcurr**
Chapter 12, "Updating a Record"

isrewrite

Rewrite a record identified by its primary key.

Syntax:

```
isrewrite(isfd, record)
int             isfd;
char            *record;
```

Description:

The **isrewrite** function rewrites a record specified by its primary key when the primary key does not allow duplicate values. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file containing the record to be updated.
record	Character buffer whose contents are written into the file. The primary key field identifies the record to be updated.

The record buffer should be loaded with the new data before the **isrewrite** call is issued. The primary key field must match that of the record to be updated. Care must be taken to ensure that fields are located properly in the buffer.

Each index defined on the data file, including the primary index, is updated as needed.

On return, **isrecnum** is set to the rewritten record, and the current record position is unchanged.

On success, **isrewrite** returns a positive integer. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
isrewrite( bookfd, buffer ) ;
```

See Also: **isrewrec**, **isrewcurr**
Chapter 12, "Updating a Record"

issetunique

Set the value of a unique identifier.

Syntax:

```
issetunique (isfd, uniqueid)
int         isfd;
long        uniqueid
```

Description:

The **issetunique** routine sets the value of a stored unique identifier to the one specified. The identifier can then be incremented as desired. The unique identifier is a long contained in **uniqueid.**

You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor of an open ISAM file.
uniqueid	The long integer specifying the unique identifier.

On success, **issetunique** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

See Also: **isuniqueid**

isstart

Select an index and locate a record.

Syntax:

```
isstart(isfd, keydesc, length, record, mode)
int             isfd;
struct keydesc  *keydesc;
int             length;
char            *record;
int             mode;
```

Description:

The **isstart** function selects an index for subsequent use. It also locates, but does not read, a record based on the selected index. Use the **isread** function to read this and subsequent records using this index.

You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file that has the index to be selected.
keydesc	Structure describing the index to be selected. The index must already exist.
length	Number of bytes of the index key to use during the search for the record. Zero indicates that the whole key is to be used.
record	Character buffer holding the key value to be used in locating the record if mode is one of ISEQUAL, ISGREAT, or ISGTEQ.
mode	Integer that determines the logic for searching the index for the desired record. The allowed values are described below.

If the mode is set to ISEQUAL, ISGREAT, or ISGTEQ, then prior to calling the function a key value is placed in the record buffer. The value's position in the record buffer must match the position specified in **keydesc**. The length parameter determines how many bytes of the key are used in the search.

The mode parameter tells **isstart** how to find the desired record. The allowed modes as defined in *isam.h* are:

Start Mode	Description
ISFIRST	first record
ISLAST	last record
ISEQUAL	first exact match of search value
ISGREAT	first record greater than search value
ISGTEQ	first record greater than or equal to search value

If the desired record is known to be the first or last in the current ordering, then use the ISFIRST or ISLAST mode value. In these cases, the record and length parameters are not used.

If ISFIRST is used, then the pointer is positioned just before the first record in the selected index order. You should then call the **isread** function using the ISNEXT mode parameter to read the first record in the selected order.

If ISLAST is used, then the pointer is positioned just after the last record in the selected index order. You should then call the **isread** function using the ISPREV mode parameter to read the last record in the selected order.

If the position of the desired record is not first or last, then use one of the ISEQUAL, ISGREAT, or ISGTEQ mode values (and a key value in record) to locate the desired record.

If ISEQUAL is used, the function locates the first record with a key value matching the record parameter.

If ISGREAT is used, the function locates the first record whose key value is greater than the key field in the record parameter.

If ISGTEQ is used, the function locates the first record whose key value is greater than or equal to the key field in the record parameter.

A special case of **isstart** can be used to access a file in physical order. To do this, set the value of the k_nparts member of **keydesc** to zero. **isstart** ignores all indexes and position on the first physical record. See **isread** for more information.

If a match is found, then the current record position pointer and **isrecnum** are set to this record. If no match is found, then **iserrno** is set to ENOREC and a -1 error code is returned.

Example:

This example makes *titlekey* the current index, uses the entire key, and locates a record matching the search value in the buffer:

```
isstart( bookfd, titlekey, 0, buffer, ISEQUAL ) ;
```

See Also: **isread**

isuniqueid

Return a unique identifier.

Syntax:

```
isuniqueid (isfd, uniqueid)
int        isfd;
long       *uniqueid
```

Description:

The **isuniqueid** routine generates a unique identifier for an ISAM file previously opened by **isopen** or **isbuild**, and identified by **isfd**. The unique identifier is a pointer to a long integer contained in **uniqueid**.

You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor of an open ISAM file.
uniqueid	Pointer to a long integer to receive the unique identifier.

On success, **isuniqueid** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

See Also: **issetunique**

isunlock

Unlock an ISAM file.

Syntax: **isunlock(isfd)**
int isfd;

Description:

The **isunlock** function unlocks a file that was locked with **islock** (the file must have been opened using the ISMANULOCK value of the mode parameter in **isopen** or **isbuild**)

You can use the following argument with this routine:.

Argument	Description
isfd	File descriptor identifying the ISAM file to be unlocked.

On success, **isunlock** returns 0. Otherwise, it returns -1 and the error is set to **iserrno**. Once you use **isunlock**, other processes may read and write into the file.

Example:

```
isunlock( bookfd ) ;
```

See Also: **isopen, islock, isclose**
Chapter 14, "Locking"

iswrcurr

Write a new record and make it current.

Syntax:

```
iswrcurr(isfd, record)
int      isfd;
char     *record;
```

Description:

The **iswrcurr** function writes a new record into an ISAM file. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file in which the record is being written.
record	Character buffer holding the data to be written into the file.

When you use **iswrcurr**, care must be taken to ensure values are located properly in their fields in the record buffer prior to calling the function.

If the function executes successfully, the current record position pointer and **isrecnum** are both set to the new record.

Any indexes are updated as needed.

If the record contains any duplicate values in fields that are defined with NODUPS attribute, an error is produced and no record is written.

On success, **iswrcurr** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
iswrcurr( bookfd, buffer ) ;
```

See Also: **iswrite**
Chapter 9, "Adding Data"

iswrite

Write a new record into an ISAM file.

Syntax:

```
iswrite( isfd, record )
int          isfd;
char         *record;
```

Description:

The **iswrite** function writes a new record into an ISAM file. You can use the following arguments with this routine:

Argument	Description
isfd	File descriptor identifying the ISAM file in which the record is being written.
record	Character buffer holding the data to be written into the file.

When using **iswrite**, care must be taken to ensure values are located properly in their fields in the record buffer prior to calling the function.

If the function executes successfully, **isrecnum** is set to the new record. The current record position pointer is left unchanged.

Any indexes are updated as needed.

If the record contains any duplicate values in fields that are defined with NODUPS attribute, an error is produced and no record is written.

On success, **iswrite** returns 0. Otherwise, it returns -1 and sets **iserrno** to indicate the error.

Example:

```
iswrite( bookfd, buffer ) ;
```

See Also: **iswrcurr**
Chapter 9, "Adding Data"

The isam.h Header File

This appendix lists the contents of the *isam.h* header file. This file holds the definitions of structures and macros used by the ISAM functions. This file is necessary for the proper functioning of calls in the *isam* library.

To include this file in your programs, insert this line near the beginning of each program file:

```
#include <isam.h>
```

You can also refer to this listing of the *isam.h* file when you need to know the value of a #define macro, the meaning of an error code, or the structure of an index key. This is particularly useful when examining the examples in the preceding chapters.

```
#ifdef M_I386
#pragma pack(2)
#endif

#define CHARTYPE     0
#define CHARSIZE     1

#define INTTYPE      1
#define INTSIZE      2

#define LONGTYPE     2
#define LONGSIZE     4

#define DOUBLETYPE   3
#define DOUBLESIZE   (sizeof(double))

#define FLOATTYPE    4
#define FLOATSIZE    (sizeof(float))
```

```
#define MAXTYPE      5

#define ISDESC       0x80     /* descending order */
#define ISNOCASE     0x100    /* no case sensitivity */

#define TYPEMASK     0x7F     /* mask for key type */

#define BYTEMASK     0xFF     /* mask for one byte */
#define BYTESHFT     8        /* shift for one byte */

#define ldint(p)     ( * (short *) (p) )
#define stint(i,p)   ( *( (short *) (p) ) = i )

long         ldlong () ;
double       lddbl () ;
float        ldfloat () ;

#define ISFIRST      0        /* position to first record*/
#define ISLAST       1        /* position to last record*/
#define ISNEXT       2        /* position to next record*/
#define ISPREV       3        /* position to previous record*/
#define ISCURR       4        /* position to current record*/
#define ISEQUAL      5        /* position to equal value*/
#define ISGREAT      6        /* position to greater value*/
#define ISGTEQ       7        /* position to = value   */

/* isread lock modes */
#define ISLOCK       0x100    /* lock record before reading*/

/* isopen, isbuild lock modes */
#define ISAUTOLOCK   0x200    /* automatic record lock */
#define ISMANULOCK   0x400    /* manual record lock*/
#define ISEXCLLOCK   0x800    /* exclusive isam file lock*/

#define ISINPUT      0        /* open for input only*/
#define ISOUTPUT     1        /* open for output only*/
#define ISINOUT      2        /* open for input and output*/


#define MAXKEYSIZE   120      /* Max number ofbytes in key*/

#define NPARTS       9        /* max number of key parts*/

struct keypart {
short kp_start ;              /* starting byte of key part*/
short kp_leng ;               /* length in bytes*/
short kp_type ;               /* type of key part*/
};
```

```
struct keydesc {
    short   k_flags ;       /* flags                    */
    short   k_nparts ;      /* number of parts in key*/
    struct  keypart
            k_part[NPARTS] ; /* each key part */

};

#define k_start   k_part[0].kp_start
#define k_leng    k_part[0].kp_leng
#define k_type    k_part[0].kp_type

#define ISNODUPS  000       /* no duplicates allowed */
#define ISDUPS    001       /* duplicates allowed*/
#define DCOMPRESS 002       /* duplicate compression */
#define LCOMPRESS 004       /* leading compression*/
#define TCOMPRESS 010       /* trailing compression*/
#define COMPRESS  016       /* all compression*/
#define ISALL     017       /* set all flags */

struct dictinfo {
short di_nkeys ;            /* number of keys defined*/
short di_recsize ;          /* data record size*/
short di_idxsize ;          /* index record size*/
long di_nrecords ;          /* number of records in file*/
};

#define EDUPL     100       /* duplicate record*/
#define ENOTOPEN  101       /* file not open */
#define EBADARG   102       /* illegal argument*/
#define EBADKEY   103       /* illegal key desc*/
#define ETOOMANY  104       /* too many files open*/
#define EBADFILE  105       /* bad isam file format*/
#define ENOTEXCL  106       /* non-exclusive access*/
#define ELOCKED   107       /* record locked */
#define EKEXISTS  108       /* key already exists*/
#define EPRIMKEY  109       /* is primary key*/
#define EENDFILE  110       /* end/begin of file*/
#define ENOREC    111       /* no record found*/
#define ENOCURR   112       /* no current record*/
#define EFLOCKED  113       /* file locked*/
#define EFNAME    114       /* file name too long*/
#define ENOLOK    115       /* can't create lock file */
#define EBADMEM   116       /* can't alloc memory*/
#define EBADCOLL  117       /* bad custom collating */
#define EPKDUP    118       /* duplicate primary key allowed */
```

```
extern int iserrno ;                    /* isam error return code */
extern char isstat1, isstat2 ;          /* COBOL return codes */
extern long isrecnum ; /* record position of the last read or write */


#ifdef M_I386
#pragma pack()
#endif
```

The example.h Header File

This appendix lists the contents of the *example.h* header file. This file contains defines and macros that are used by several of the example programs in this book.

```
/*      example.h
 *      Header file describing the structure of "books" and "authors"
 *      books file
 *      field           type    offset  length  comments
 *      ------------------------------------------------------------
 *      bookcode        long    0       4       Primary key unique
 *      title           char    4       40
 *      publisher       char    44      30
 *      quantity        long    74      4
 *      price           float   78      4
 *
 *      authors file
 *      field           type    offset          length  comments
 *      ------------------------------------------------------------
 *      bookcode long   0               4       Primary key dups
 *      lname           char    4       20      Alternate key part 1
 *      fname           char    24      20      Alternate key part 2
 *
 *      bookcode is the key that connects both books and authors. A
 *      single book may have multiple authors. The authors file allows
 *      duplicate occurences of bookcode.
 *      The notation used in naming is suffix of OFF for offset and SIZE
 *      for length. The first byte in a record is at offset 0.
 *      Note:
 *      Many constants defined in this file assume that isam.h is already
 *      included */
```

```
/* books file size and offset definitions */

#define BOOKCDSIZE      LONGSIZE
#define TITLESIZE       40
#define PUBLSIZE        30
#define QTYSIZE         LONGSIZE
#define PRICESIZE       FLOATSIZE
#define LNAMESIZE       20 #define FNAMESIZE     20
#define BOOKRECSIZE     (BOOKCDSIZE+TITLESIZE+PUBLSIZE+QTYSIZE+PRICESIZE)
#define AUTHRECSIZE     BOOKCDSIZE + LNAMESIZE+FNAMESIZE


/* books and author field positions       */

#define  BOOKCDOFF      0
#define  TITLEOFF       BOOKCDOFF + BOOKCDSI
#define  PUBLOFF        TITLEOFF + TITLESI
#define  QTYOFF         PUBLOFF + PUBLSI
#define  PRICEOFF       QTYOFF + QTYSI
#define  LNAMEOF        BOOKCDOFF + BOOKCDSIZ
#define  FNAMEOFF       LNAMEOFF + LNAMESIZE
```

C

Exception Handling

It is generally the responsibility of the programmer to handle error conditions that arise as a result of ISAM function calls. To facilitate this task, a set of error codes is provided to indicate the nature of each error. In addition, diagnostics about the type of error are provided by two global status characters, **isstat1** and **isstat2**. The **isstat1** variable indicates general information about the status of a function call. The **isstat2** variable provides further information, based on the value of **isstat1**.

ISAM Error Codes

Table B-1 provides the constant names for error values as defined in *isam.h*, numerical values of **iserrno**, and descriptions of ISAM errors.

Table B-1. ISAM Error Codes

Constant	iserrno	Description	isstat1	isstat2
EDUPL	100	Duplicate. The call tried to add a duplicate value to an index where duplicates are not permitted. The call was one of **isaddindex**, **isrewcurr**, **isrewrite**, or **iswrite**.	2	2
ENOTOPEN	101	Not open. The call tried to operate on a file that was not yet opened or tried to perform an operation not specified by the **isopen** read/ write mode value in effect.	9	0

(Continued on next page.)

Table B-1. ISAM Error Codes *(Continued)*

Constant	iserrno	Description	isstat1	isstat2
EBADARG	102	Bad argument. One of the call's arguments is outside the range of allowable values for that argument.	9	0
EBADKEY	103	Bad key. At least one of the elements of an index key description is not within the range of allowable values for that element.	9	0
ETOOMANY	104	Too many. The call attempted to exceed the maximum number of files (20) that can be open at one time.	9	0
EBADFILE	105	Bad file. The format of the file has been damaged.	9	0
ENOTEXCL	106	Not exclusive. The call tried to add or delete an index but the file had not been opened in exclusive mode.	9	0
ELOCKED	107	Locked. The call requested a record that has been locked by another process.	9	0
EKEXISTS	108	Key exists. The call tried to add an index that already exists.	9	0

(Continued on next page.)

Table B-1. ISAM Error Codes *(Continued)*

Constant	iserrno	Description	isstat1	isstat2
EPRIMEKEY	109	Primary key. The call tried to delete the primary key index. The primary key may not be deleted with **isdelindex**.	9	0
EENDFILE	110	End of file. The beginning or end of the file was encountered.	1	0
ENOREC	111	No record. The file contains no record that matches the search conditions.	2	3
ENOCURR	112	No current record. This call must operate on the current record, but not current record has been defined.	2	1
EFLOCKED	113	File locked. The file is exclusively locked by another process.	9	0
EFNAME	114	Filename. The filename being specified is too long.	9	0
ENOLOK	115	No lock. Unable to establish lock manager. Check the DBKEY environment variable.		
EBADMEM	116	Bad memory. The call was unable to allocate sufficient memory to continue.		

(Continued on next page.)

Table B-1. ISAM Error Codes *(Continued)*

Constant	iserrno	Description	isstat1	isstat2
EBADCOLL	117	Bad collating.		
EPKDUP	118	Primary key duplicate.		

The isstat1 and isstat2 Status Characters

Table B-2 lists the **isstat1** status character codes that provide additional information about certain ISAM conditions.

Table B-2. isstat1 Status Character Values

isstat1	Description
0	Successful execution of the call.
1	End of file was encountered.
2	Invalid key.
3	System error.
9	User-defined error.

Table B-3 lists the **isstat1** codes and the corresponding **isstat2** codes.

Table B-3. isstat2 Status Character Values

isstat1	isstat2	Description
0-9	0	No more information is available.
0	2	Duplicate key. If the call was isread, this indicates the key value in the next record equals that of the current record. If the call was **iswrite** or **isrewrite**, this indicates the new record created a duplicate key in at least one alternate index that allows duplicates.
2	1	The primary key value of the record was changed between the **isread** call and the **isrewrite** call.
2	2	The call tried to add a duplicate value to an index where duplicates are not permitted. The call was one of **isaddindex, isrewcurr, isrewrite,** or **iswrite.**
2		The file contains no record that matches the search conditions.
2	4	The call tried to write beyond the externally defined boundaries of the file.
9	any	User-defined values. Put the single character definitions in the *isam.h* header file.

D

Operating System Considerations

This appendix briefly describes the shared-memory environment used by SCO ISAM, including the DBKEY environment variable. It also describes three utilities that can be used to maintain and repair inconsistencies in ISAM files.

To avoid operating system dependent limitations on the number of files that can be locked at one time, SCO ISAM uses a shared-memory segment to contain its lock tables. Shared memory also allows record-level locking.

Using the DBKEY Environment Variable

SCO ISAM uses key values to identify shared-memory segments for locked ISAM files and records. The DBKEY environment variable is used to establish a common key value for users who access the same ISAM files. The value assigned to DBKEY is an integer and when multiple users share ISAM files it is important that each uses the same value.

To set the value of DBKEY:

- If you use the Bourne shell, include two lines in each user's *.profile* file:

```
DBKEY=value
export DBKEY
```

Here *value* is the value to be assigned to DBKEY. There are no spaces around the equals sign.

- If you use the C shell, include this line in each user's *.login* file:

```
setenv  DBKEY value
```

Removing Shared Memory and Semaphores

Normally when ISAM files are closed, all pertinent shared-memory segments are removed. However, if files are closed abnormally, such as when an operating system process is killed improperly, spurious active shared memory segments and semaphores may result. This may cause "record lock" errors on records that are not locked. Two utilities are distributed with the operating system for cleanup of these shared memory segments: **ipcs** and **ipcrm**.

To remove active shared-memory segments and semaphores, first use the **ipcs** command to see if this is the cause of the record lock problem. Then, use the **ipcrm** command to remove the appropriate shared memory segments and semaphores.

The procedure is as follows:

1. Make sure no one on the system is using SCO ISAM with the DBKEY corresponding to the records that produced the error. If SCO ISAM is in use, the **ipcs** command displays the active semaphores and shared memory segments for those active users as well as for the defunct processes. When no one is using SCO ISAM with a particular DBKEY, you can assume that all shared memory and semaphores with that key can be safely removed.

2. Use the **ipcs** command to display active semaphores and shared memory segments. At the system prompt, type:

   ipcs

   Your screen displays active semaphores and shared memory segments, similar to the following example:

```
IPC status from /dev/kmem as of Mon Nov 20 10:33:51 1989
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
Shared Memory (5.0):
m      1 0x00000001 --rw-rw-rw-   rogerw      dbases
m      2 0x000032c8 --rw-rw-rw-   johnla      dbases
m      3 0x000106a6 --rw-rw-rw-   elinor      dbases
Shared Memory (3.0):
Semaphores (5.0):
s    100 0x00000001 --ra-ra-ra-   rogerw      dbases
s     51 0x000032c8 --ra-ra-ra-   johnla      dbases
s     32 0x000106a6 --ra-ra-ra-   elinor      dbases
Semaphores (3.0):
```

   The first column displays either an "m" (shared segment) or an "s" (semaphore). The second column (ID) displays an ID number for the shared segment or semaphore, and the third column (KEY) displays the hex value of the corresponding DBKEY.

3. Use the **ipcrm** command to remove all shared memory segments and semaphores with the DBKEY corresponding to the records that produced the error. The syntax is as follows:

   ipcrm -(**m** | **s**)<*ID number*>

The *ID number* is taken from the second column of the **ipcs** display. Use "m" or "s" depending on whether the ID number indicates a shared memory segment or a semaphore, respectively.

For the previous example, type the following:

ipcrm -m1 -m2 -m3 -s100 -s51 -s32

See your operating system user's reference for more information about the **ipcs** and **ipcrm** commands.

Verifying Data from Tables

Each SCO ISAM file consists of a physical data (*.dat*) file and an index (*.idx*) file. Use the **verify** utility to detect and, if specified, to repair inconsistencies between these two files. The **verify** utility checks that every valid record in the data file is properly represented in the index file; it also checks that every index entry points to a valid data record.

You must be in the database directory before you run this utility. The following is the syntax for using **verify**:

verify [**-ilpyn**] *filelist*

Replace *filelist* with the list of ISAM files to be checked by **verify**. The *.dat* and *.idx* suffixes should not be included in the *filelist*.

You can specify any of the following flags when invoking **verify**:

Flag	Description
-i	Check only indexes. This flag causes **verify** to check only the index file (as opposed to checking both the index and the data files) for consistency. Use this flag as a quick check if you think the data files are probably not corrupted.
-l	Long listing. This flag causes **verify** to print a long listing of the information for each defined key (index), along with the associated data record pointer. The key value for each data record is displayed by key part, along with the byte position of the data record in the data file.
-p	Pause after each key. This flaug causes **verify** to pause after displaying information about each index. You must press the <Return> key before the process continues.
-y	Yes, correct errors if found. This flag causes **verify** to assume a "yes" answer to each error state and to attempt to make the specified correction. When you use this flag, **verify** ceases to be interactive.
-n	No, do not correct errors if found. This flag cuases **verify** to assume a "no" answer to each error state and to leave the files unchanged. It also allows you to see where errors are by displaying them on the screen. When you use this flag, **verify** ceases to be interactive.

It is recommended that you use **verify** with the **-y** flag so that the utility attempts to correct any discrepancies automatically.

Whether or not you use **verify** with the **-l** or **-p** flags, if an error is detected, you have the option of making a correction or leaving the files unchanged. If no errors are detected, no response is required. If you choose to make a correction, **verify** attempts to repair the files. Unless the **-y** or **-n** flags are specified on the command line, you must choose interactively whether or not to make each correction.

If any file associated with the table is corrupted beyond repair, **verify** quits and displays an error message.

Index

K

L

M

N

O

P

R

S

U

W